Visual LANSA Feature Help

This file contains text that responds as help when you press F2 on a Visual LANSA event or component or press the question mark icon on the Feature tab's toolbar.

The text is not intended to be read as a guide but has been provided so that it is selected, if appropriate, in response to a search of the LANSA knowledge database.

Edition Date: October 28, 2014 © LANSA **Control and Composite**

Origin

The variables specified in the Origin selector will contain a reference the control on which the event was first fired.

Handled

The Handled selector allows a user to stop routed events going any further along the parent chain. The default is false

True

The event is being processed in this event routine and should not be routed to the parent.

False

The event is not being fully processed in this event routine and should be routed to the parent.

FadeIn Method

DirectX Only

FadeIn is an animation method that transitions a control's Opacity property from its current value to 100 over the specified duration. When the animation completes the Visible property will be True.

Like all animations, it is executed in a separate thread allowing other processing to continue.

Delay

Number of milliseconds to delay the processing of the animation. The default is 0.

Duration

Number of milliseconds required to complete the animation. Default is 250.

FadeOut Method

DirectX Only

FadeOut is an animation method that transitions a control's Opacity property from its current value to 0 over the specified duration. When the animation completes the Visible property will be False.

Like all animations, it is executed in a separate thread allowing other processing to continue.

Delay

Number of milliseconds to delay the processing of the animation. The default is 0.

Duration

Number of milliseconds required to complete the animation. Default is 250.

Scale Method

DirectX Only

Scale is an animation method that transitions a control's ScaleWidth and ScaleHeight properties from their current value to the specified value over the specified duration. When the animation completes the control's ScaleHeight and ScaleWidth properties will match the specified values.

Like all animations, it is executed in a separate thread allowing other processing to continue.

ScaleWidth

Target ScaleWidth value for the animation. Default is 100.

ScaleHeight

Target ScaleHeight value for the animation. Default is 100.

Delay

Number of milliseconds to delay the processing of the animation. The default is 0.

Duration

Number of milliseconds required to complete the animation. Default is 250.

MouseEnter Event

DirectX Only

MouseEnter is fired when the mouse enters the outer bounds of a control.

MouseHover Event

DirectX Only

MouseHover is fired when the mouse is stationary over a control for a predetermined amount of time. The event is fired once and will only fire again when the mouse has left the re-entered the control.

MouseLeave Event

DirectX Only

MouseLeave is fired when the mouse leaves the outer bounds of a control. It is not fired on the parent when the mouse enters the bounds of a child control.

Hint Popup Property

DirectX Only

Hint Popup allows a PopupPanel (#prim_ppnl) instance to be used instead of the default popup text box. This gives the developer complete control over the content and format of the hint.

Define_Com Class(#prim_phbn) Name(#Button) Caption('Click') Hintpopup(#ButtonHintPanel) Define_Com Class(#Prim_ppnl) Name(#ButtonHintPanel) Content(#MyHintPanelContent)

Hint Title Property

DirectX Only

When specified, the HintTitle is shown at the top of the hint window in emphasized text. The remainder fof the hint is shown below as normal.

IsAnimating Property

DirectX Only

Used to determine whether a control is being animated. Typically used to prevent an animation being restarted.

True

The control is not being animated

False

The control is being animated

MouseOver Property

DirectX Only

Determines whether the mouse is directly over a control. If the mouse is over a child control, MouseOver will be false.

MouseOver is useful when MouseEnter/Leave code is being run to change the appearance of a control, but other factors are also affecting the control's state.

True

The mouse is directly over the control

False

The mouse is not directly over the control

MouseOverStyle Property

DirectX Only

Denotes the Style (#Prim_vs.Style) to be applied to a control in the event of the Mouse entering the physical bounds of the control. The style is removed when the mouse leaves the control.

MouseOverStyle negates the need to code many MouseEnter and corresponding MouseLeave events and allows for simple declaration instead.

MouseOverStyles Property

DirectX Only

Collection of Styles (#Prim_vs.Style) to be applied to a control in the event of the Mouse entering the physical bounds of the control. The styles are removed when the mouse leaves the control.

MouseOverStyles allows for more complex programmatic appearance changes to be coded. Rather than relying on the declarative single MouseOverStyle, the developer is free to add as many style layers as required.

Opacity Property

DirectX Only

Opacity denotes the appearance of a control in terms of its interaction with its background.

The default is 100; entirely opaque. No background will be visible at all.

As the value decreases, more and more of the background will become visible through the control. When the value is 0 the control will be entirely transparent and only the background will be visible.

Note: a control with opacity of 0 is not visible to the user but will continue to function as normal and is still both visible and enabled.

Popup Property

DirectX Only

Popup allows a PopupPanel (#prim_ppnl) instance to be used instead of the typical right click popup menu (Prim_pmnu). This gives the developer complete control over the content and format of the popup.

Define_Com Class(#prim_trvw) Name(#Tree) Popup(#TreePopupPanel) Define_Com Class(#Prim_ppnl) Name(#TreePopupPanel) Content(#MyTreePopupPanel)

Rotation Property

DirectX Only

Rotation allows a control to rotate about a given origin as specified by the RotationOriginLeft and RotationOriginTop properties.

Rotation is a measured in degrees and has a range of values from 0 to 359.

Rotation, like Scale and Skew, is a visual effect and does not alter the fundamental size and position of the control. The left, top, height and width properties maintain their value. Rotated controls are clipped to ensure they fit within the bounds of their parent control.

RotationOriginLeft Property

DirectX Only

In conjunction with RotationOriginTop, defines the coordinates of an imaginary point about which a control will rotate when the Rotation property is applied. RotationOriginLeft is a percentage and has a default of 50.

RotationOriginTop Property

DirectX Only

In conjunction with RotationOriginTop, defines the coordinates of an imaginary point about which a control will rotate when the Rotation property is applied. RotationOriginTop is a percentage and has a default of 50.

ScaleHeight Property

DirectX Only

In conjunction with ScaleWidth, defines the visible size of the control based on an imaginary point defined by the ScaleOriginLeft and ScaleOriginTop properties.

ScaleHeight is a percentage. The default is 100.

Scale, like Rotation and Skew, is a visual effect and does not alter the fundamental size and position of the control. The left, top, height and width properties maintain their value. Scaled controls are clipped to ensure they fit within the bounds of their parent control.

ScaleWidth Property

DirectX Only

In conjunction with ScaleHeight, defines the visible size of the control based on an imaginary point defined by the ScaleOriginLeft and ScaleOriginTop properties.

ScaleWidth is a percentage. The default is 100.

Scale, like Rotation and Skew, is a visual effect and does not alter the fundamental size and position of the control. The left, top, height and width properties maintain their value. Scaled controls are clipped to ensure they fit within the bounds of their parent control.

ScaleOriginLeft Property

DirectX Only

In conjunction with ScaleOriginTop, defines the coordinates of an imaginary point from which a control will scale when the ScaleWidth and ScaleHeight properties are applied.

ScaleOriginLeft is a percentage and has a default of 50.

ScaleOriginTop Property

DirectX Only

In conjunction with ScaleOriginLeft, defines the coordinates of an imaginary point from which a control will scale when the ScaleWidth and ScaleHeight properties are applied.

ScaleOriginLeft is a percentage and has a default of 50.

SkewLeft Property

DirectX Only

In conjunction with SkewTop, defines the appearance of the control as though rotated around an imaginary point defined by the SkewOriginLeft and SkewOriginTop properties.

SkewLeft is measured in degrees. The default is 0.

Skew, like Rotation and Scale, is a visual effect and does not alter the fundamental size and position of the control. The left, top, height and width properties maintain their value. Skewed controls are clipped to ensure they fit within the bounds of their parent control.

SkewTop Property

DirectX Only

In conjunction with SkewLeft, defines the appearance of the control as though rotated around an imaginary point defined by the SkewOriginLeft and SkewOriginTop properties.

SkewTop is measured in degrees. The default is 0.

Skew, like Rotation and Scale, is a visual effect and does not alter the fundamental size and position of the control. The left, top, height and width properties maintain their value. Skewed controls are clipped to ensure they fit within the bounds of their parent control.

SkewOriginLeft Property

DirectX Only

In conjunction with SkewOriginTop, defines an imaginary point about which a control can be skewed using the SkewLeft and SkewTop properties.

SkewOriginLeft is a percentage. The default is 50.
SkewOriginTop Property

DirectX Only

In conjunction with SkewOriginLeft, defines an imaginary point about which a control can be skewed using the SkewLeft and SkewTop properties.

SkewOriginTop is a percentage. The default is 50.

Style Property

DirectX Only

Denotes the Style (#Prim_vs.Style) to be applied to a control. When the Style property is set, all Styles previously applied to the control are removed. Styles from parent controls will be inherited dependent on value of the

VisualStyleOfParent property.

Styles Property

DirectX Only

Collection of Styles (#Prim_vs.Style) to be applied to a control.

Styles allows for more complex programmatic appearance changes to be coded. Rather than relying on the declarative single Style property, the developer is free to add as many Style layers as required.

Styles Collection

DirectX Only

Collection of Styles (#Prim_Vcol<#Prim_vs.Style>) to be applied to a control. Styles collections allow for multiple styles to be added to or removed from a control at run time.

Style

Style (#Prim_VS.Style) reference to be added to the Styles collection. If the Style being added to the collection is already in the collection, the Add

request is ignored.

Add Method

Add a Style to the Styles collection

If the Style being added to the collection is already in the collection, the Add request is ignored.

Remove Method

Remove a Style from the Styles collection

If the Style to be remove from the collection is not in the collection, the Remove request is ignored.

RemoveAll Method

Remove all Styles from the Styles collection

MouseOverPart Property

DirectX Only

Determines whether the mouse is directly over a control or any child control. In effect, this determines whether the mouse is within the physical bounds of the control.

MouseOverPart is useful when MouseEnter/Leave code is being run to change the appearance of a control, but other factors are also affecting the control's state.

True

The mouse is directly over the control or one of its children

False

The mouse is not over the control or any of its children

PrivateStyle Property

DirectX Only

Denotes the Style (#Prim_vs.Style) to be applied to this control only.

Styles are normally inherited by child controls, depending on the VisualStyleOfParent property. PrivateStyle allows composite controls to have their own styles without affecting the appearance of child controls.

For example, a group box may need to have a bold text but normally, applying bold to a composite directly will cause the both the caption and contents to become bold. By specifying this as a Private Style the group box can step out of the style inheritance chain.

PrivateStyles Property

DirectX Only

Collection of Styles (#Prim_vs.Style) to be applied to this control only. Styles are normally inherited by child controls, depending on the VisualStyleOfParent property. PrivateStyle allows composite controls to have their own styles without affecting the appearance of child controls. Styles collections allow for multiple styles to be added to or removed from a control at run time.

Transition Method

DirectX Only

Transition is a method that animates the change from one component to another.

The controls specified in the To and From parameters are switched from Visible(True) to Visible(False) respectively. The style of animation is dependent on the TransitionType property

For the best results, the two controls should occupy the same physical bounds.

Like all animations, it is executed in a separate thread allowing other processing to continue.

If the animation is still running when the method is called again, the animation will reset and begin again.

То

Reference to a Control (#Prim_Ctrl).

The control must be parented to the composite. When the animation complete the control will be Visible(True)

From

Reference to a Control (#Prim_Ctrl).

The control must be parented to the composite. When the animation completes the control will be Visible(False)

TransitionType

There are a number of transition types. The default is fade.

Delay

Number of milliseconds to delay the processing of the animation. The default is 0.

Duration

Number of milliseconds required to complete the animation. Default is 250.

RenderStyle

In conjunction with the RNDR X_RUN argument, the RenderStyle properties of Form (#Prim_FORM) and Panel (#Prim_PANL) determine how some or all of the application is rendered.

The RNDR runtime argument has 3 possible values: W, M and X. Win32, Win32 and DirectX, and DirectX. For any of the application to render in DirectX, the runtime argument must be either M or X.

If the value is W, DirectX will not be used regardless of the value of the properties specified.

Form (#Prim_Form) has 3 possible values:

ApplicationThe RNDR runtime argument value will be used. This is the default.

DirectX

The form will use DirectX if the RNDR runtime argument is either M or X.

Win32

The form will be Win32 regardless of the runtime setting.

Panel (#Prim_Panl) has 2 possible values:

Note that there is no option for a panel to specifically be Win32. Once a panel is DirectX, all child panels must be DirectX.

DirectX

The panel will use DirectX if the RNDR runtime argument is either M or X.

Parent

The panel will render dependent on its parent's RenderStyle. This is the default.

The defaults of Application for Form and Parent for panels allows DirectX to be implemented throughout the application with a change to a single runtime setting.

If you only want to enable DirectX in a piecemeal fashion, perhaps for a few panels so that you can incorporate new DirectX controls and styles, change the runtime setting to M so that DirectX is allowed and then change the forms or panels to be DirectX as required.

Style (Prim_Vs.Style)

Styles are used to change the appearance of a control. They allow you to define the color of text, background colors, fonts, borders and other visual features.

Unlike Visual Styles, which are mutually exclusive, multiple Styles can be added to a control affecting only the features specified on the style. If the same feature of a style is applied using two different style instances, the last one to be applied will take precedence.

Only foreground features are adopted by child controls e.g. Facename and TextColor.

BackgroundBrush Property

DirectX Only

Reference to an instance of a Brush (#Prim_vs.Brush) applied to the background of the control.

Background features of a Style are not inherited by child controls. This ensures that a background applied to a form is not repeated verbatim on child controls.

If both a BackgroundBrush and NormBackColor are specified, the BackgroundBrush will take precedence.

The BackgroundBrush property can be used as shown in the following Example.

Define_Com Class(#PRIM_VS.Style) Name(#BackGround) Backgroundbrush(#Backgroundbrush) Define_Com Class(#PRIM_VS.LinearBrush) Name(#Backgroundbrush) Colors(#BackgroundbrushColors) Define_Com Class(#Prim_Vs.BrushColors) Name(#BackgroundbrushColors) Define_Com Class(#PRIM_VS.BrushColor) Name(#BackgroundbrushColor1) Color(Silver) Parent(#BackgroundbrushColors) Define_Com Class(#PRIM_VS.BrushColor) Name(#BackgroundbrushColor1) At(100) Color(White) Parent(#BackgroundbrushColors)

BorderBottom Property

DirectX Only

In conjunction with BorderTop, BorderLeft and BorderRight, BorderBottom defines the thickness of the border displayed around a control.

Borders consume part of the control when applied. This is particularly relevant when applying a style with top and left borders on MouseOver. The top and left borders will effectively cause the 0, 0 coordinates to move resulting in child control moving screen position.

BorderBrush Property

DirectX Only

Reference to an instance of a Brush (#Prim_vs.Brush) applied to the border of the control.

BorderLeft Property

DirectX Only

In conjunction with BorderTop, BorderBottom and BorderRight, BorderLeft defines the thickness of the border displayed around a control.

Borders consume part of the control when applied. This is particularly relevant when applying a style with top and left borders on MouseOver. The top and left borders will effectively cause the 0, 0 coordinates to move resulting in child control moving screen position.

BorderRight Property

DirectX Only

In conjunction with BorderTop, BorderLeft and BorderBottom, BorderRight defines the thickness of the border displayed around a control.

Borders consume part of the control when applied. This is particularly relevant when applying a style with top and left borders on MouseOver. The top and left borders will effectively cause the 0, 0 coordinates to move resulting in child control moving screen position.

BorderTop Property

DirectX Only

In conjunction with BorderBottom, BorderLeft and BorderRight, BorderTop defines the thickness of the border displayed around a control.

Borders consume part of the control when applied. This is particularly relevant when applying a style with top and left borders on MouseOver. The top and left borders will effectively cause the 0, 0 coordinates to move resulting in child control moving screen position.

CornerBottomLeft Property

DirectX Only

In conjunction with CornerTopLeft, CornerTopRight, and CornerBottomRight, CornerBottomLeft defines the radius of the corners for a control. By default, all corners have a 0 radius and are right angles.

Unlike borders, which change the position of the 0,0 coordinate, corners are a visual effect. When applied, the background of the control is clipped and the control will appear rounded. However, foreground features are not clipped and text may extend in to corners.

CornerBottomRight Property

DirectX Only

In conjunction with CornerTopLeft, CornerTopRight, and CornerBottomLeft, CornerBottomRight defines the radius of the corners for a control. By default, all corners have a 0 radius and are right angles.

Unlike borders, which change the position of the 0,0 coordinate, corners are a visual effect. When applied, the background of the control is clipped and the control will appear rounded. However, foreground features are not clipped and text may extend in to corners.

CornerTopLeft Property

DirectX Only

In conjunction with CornerBottomLeft, CornerBottomRight and CornerTopRight, CornerTopLeft defines the radius of the corners for a control. By default, all corners have a 0 radius and are right angles.

Unlike borders, which change the position of the 0,0 coordinate, corners are a visual effect. When applied, the background of the control is clipped and the control will appear rounded. However, foreground features are not clipped and text may extend in to corners.

CornerTopRight Property

DirectX Only

In conjunction with CornerBottomLeft, CornerBottomRight and CornerTopLeft, CornerTopRight defines the radius of the corners for a control. By default, all corners have a 0 radius and are right angles.

Unlike borders, which change the position of the 0,0 coordinate, corners are a visual effect. When applied, the background of the control is clipped and the control will appear rounded. However, foreground features are not clipped and text may extend in to corners.

Effect Property

DirectX Only

Reference to an instance of an Effect (#PRIM_VS.Effect)

Effects are used to augment the features specified on the style with blurring or a drop shadow to help present a 3D effect.

ForegroundBrush Property

DirectX Only

Reference to an instance of a Brush (#Prim_vs.Brush) applied to the foreground of the control.

Foreground features of a Style are inherited by child controls. This ensures that a font applied by a style to a form will be used consistently across all child controls.

If both a ForegroundBrush and TextColor are specified, the ForegroundBrush will take precedence.

MaskBrush

Reference to an instance of a Brush (#Prim_vs.Brush) applied to a control to selectively apply transparency. This allows for controls, particularly images, to fade out and blend in to the background.

In the case of a linear brush with a gradient from any color to transparent, the control will be become increasingly transparent. If using a Visual Brush, transparency will be applied based on the transparency of the image used in the Visual Brush.

Brush

Base class for all Brush classes i.e. Solid, Linear, Radial, Image and Visual Brushes

Brushes are used by styles (#Prim_VS.Style) to define the foreground and background appearance.

See the individual Brush classes for more detail.

BrushColors

BrushColors is a collector of BrushColor (Prim_Vs.BrushColor) instances. This abstraction allows same set of colors to be used on multiple brushes.

BrushColors uses BrushColor instances to describe the transition from one color to another. Typically, this might be a simple gradient transition from silver at the top to white at the bottom, but the number of colors and the nature of the transition are dependent on the configuration of the specific class of brush being used and the definition of the color instances.

The sample below changes from blue at the start to red at the end. This is denoted by the use of "At(100)" on the second color (#Color2)

Define_Com Class(#Prim_Vs.BrushColors) Name(#Colors)

Define_Com Class(#Prim_Vs.BrushColor) Name(#Color1) Color(0:0:255) Parent(#Colors)

Define_Com Class(#Prim_Vs.BrushColor) Name(#Color2) At(100) Color(255:0:0) Parent(#Colors)

As the transition is uniform the blue channel is decremented from 255 to 0 while the red channel increases from 0 to 255. The result is that at the midpoint the color will be 128:0:128, which is purple.

BrushColor

Used to define a color as part of BrushColors (#Prim_VS.BrushColors) instance.

At Property

At determines the point on an imaginary line at which the gradient color will be shown as the color specified in the Color property.

At is a percentage.

Define_Com Class(#Prim_Vs.BrushColors) Name(#Colors) Define_Com Class(#Prim_Vs.BrushColor) Name(#Color1) Color(Silver) Parent(#Colors) Define_Com Class(#Prim_Vs.BrushColor) Name(#Color2) At(100) Color(White) Parent(#Colors)
Color Property

Denotes the color of the brush at the point determined by the At property. Color can be any valid RGB value e.g. 255:0:0 for red, or any of a series of predefined named colors such as red, blue, yellow, or transparent.

When brush colors are being used for a MaskBrush, the color property is evaluated as either Transparent or Opaque, regardless of the color specified.

The sample below changes from White to Transparent. The result is that as the transition occurs, the background will become progressively more visible.

Define_Com Class(#Prim_Vs.BrushColors) Name(#Colors) Define_Com Class(#Prim_Vs.BrushColor) Name(#Color1) Color(White) Parent(#Colors)

Define_Com Class(#Prim_Vs.BrushColor) Name(#Color2) At(100) Color(Transparent) Parent(#Colors)

Parent Property

Reference to a BrushColors (#Prim_Vs.BrushColors) used to collect individual color instances.

Gradient Brush

Brush used to define a gradient color using one or more colors.

Colors Property

Reference to a BrushColors (#Prim_VS.BrushColors) instance. Defines the colors used in the brush.

Opacity Property

Opacity denotes the appearance of a gradient in terms of its interaction with its background.

The default is 100; entirely opaque. Nothing behind the gradient will be visible at all.

As the value decreases, more and more of the background will become visible through the control. When the value is 0 the control will be entirely transparent and only the background will be visible.

When specified on a control, opacity affects the whole of the control, both foreground and background. When specified on a brush, opacity can be used to target either the foreground or the background, allowing for a semi-transparent background, but still fully opaque text.

Spread Property

Used to define the appearance of the brush beyond the extent of the brush colors or the logical start and end of the specific brush.

Pad

Any space before the start or end of the brush colors will be filled with the colors specified as the first and last colors respectively.

Thus, in the example below, where red does not start until 20% of the was across the gradient, the first 20% will be colored red. Similarly, the 20% after blue will be colored blue.

Define_Com Class(#Prim_Vs.LinearBrush) Name(#LinearBrush) Colors(#LinearBrushColors) Define_Com Class(#Prim_Vs.BrushColors) Name(#LinearBrushColors) Define_Com Class(#Prim_Vs.BrushColor) Name(#LinearBrushColor1) At(20) Color(Red) Parent(#LinearBrushColors) Define_Com Class(#Prim_Vs.BrushColor) Name(#LinearBrushColor2) At(80) Color(Blue) Parent(#LinearBrushColors)

Reflect

Any space before the start or end of the brush colors will be filled with a reflection of the brush.

Repeat

Any space before the start or end of the brush colors will be filled with a repeat of the brush.

Linear Brush

A Linear Brush is used to create a gradient color that transitions between the colors specified in the Colors (#Prim_vs.BrushColors) property.

Linear brush has start and end coordinates to create an imaginary line that defines the path the gradient will follow, allowing the gradient to be vertical, horizontal or on an angle.

EndLeft Property

Used in conjunction with the EndTop, StartLeft and StartTop properties to define the path and extent of the brush evaluation. EndTop is a percentage.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(100) EndTop(100) will appear to travel diagonally from the top left to the bottom right.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(100) will appear to travel vertically from the top to the bottom.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(50) will appear to travel vertically from the top to half way down the control. The Spread property will determine appearance beyond the extent of the start and end coordinates.

EndTop Property

Used in conjunction with the EndLeft, StartLeft and StartTop properties to define the path and extent of the brush evaluation. EndLeft is a percentage.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(100) EndTop(100) will appear to travel diagonally from the top left to the bottom right.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(100) will appear to travel vertically from the top to the bottom.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(50) will appear to travel vertically from the top to half way down the control. The Spread property will determine appearance beyond the extent of the start and end coordinates.

This collection of properties define the line along which a gradient color will change. It has StartTop and StartLeft co-ordinates, and EndTop and EndLeft co-ordinates. A StartLeft and StartTop of 0 refer to the top left hand corner of an area, and EndTop and EndLeft of 100 refern to 100% across and down, thus to the bottom right corner of the area.

StartLeft Property

Used in conjunction with the EndTop, EndLeft and StartTop properties to define the path and extent of the brush evaluation. StartLeft is a percentage.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(100) EndTop(100) will appear to travel diagonally from the top left to the bottom right.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(100) will appear to travel vertically from the top to the bottom.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(50) will appear to travel vertically from the top to half way down the control. The Spread property will determine appearance beyond the extent of the start and end coordinates.

StartTop Property

Used in conjunction with the EndTop, EndLeft and StartLeft properties to define the path and extent of the brush evaluation. StartTop is a percentage.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(100) EndTop(100) will appear to travel diagonally from the top left to the bottom right.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(100) will appear to travel vertically from the top to the bottom.

A Brush defined as StartLeft(0) StartTop(0) EndLeft(0) EndTop(50) will appear to travel vertically from the top to half way down the control. The Spread property will determine appearance beyond the extent of the start and end coordinates.

Radial Brush

A Radial Brush is used to create a gradient color that transitions between the colors specified in the Colors (#Prim_vs.BrushColors) property.

Radial Brush has Origin, Center, and Radius coordinates. The brush radiates from the Center along the extent of the Radius similar to Linear Brush (#Prim_VS.Linear Brush) but instead producing a circular pattern.

By having the Origin and Center coordinates the same, the brush will always appear as though directly in front of the view. However, changing the value changes the shape and the brush appears skewed. A simple analogy is that of looking at the beam of a torch. A RadialBrush can be thought of as a 2D view of a 3D shape. The Origin is the position of the torch, Center is the middle of the light as it shines on somthing, and Radius describes the width. If there is sufficient difference between the origin and center, it is no longer possible to see the base, just a "triangular" side view of the beam of light.

The example below is a simple radial brush that changes from red at 0, to blue producing a red center and blue outer.

Define_Com Class(#Prim_Vs.RadialBrush) Name(#RadialBrush) Colors(#RadialBrushColors)

Define_Com Class(#Prim_Vs.BrushColors) Name(#RadialBrushColors) Define_Com Class(#Prim_Vs.BrushColor) Name(#RadialBrushColor1) Color(Red) Parent(#RadialBrushColors)

Define_Com Class(#Prim_Vs.BrushColor) Name(#RadialBrushColor2) At(100) Color(Blue) Parent(#RadialBrushColors)

CenterLeft Property

In conjunction with the CenterTop property defines the start point or middle of the base of the beam. The brush colors will emanate from this point.

CenterLeft is a percentage.

Both CenterLeft and CenterTop have a default of 50.

CenterTop Property

In conjunction with the CenterTop property defines the start point or middle of the base of the beam. The brush colors will emanate from this point.

CenterTop is a percentage.

Both CenterLeft and CenterTop have a default of 50.

OriginLeft Property

In conjunction with the OriginTop property defines the position of the source. As the Origin and Center diverge the shape of the brush will tend become less circular and more splayed.

OriginLeft is a percentage.

Both OriginLeft and OriginTop have a default of 50.

OriginTop Property

In conjunction with the OriginLeft property defines the position of the source. As the Origin and Center diverge the shape of the brush will tend become less circular and more splayed.

OriginTop is a percentage.

Both OriginLeft and OriginTop have a default of 50.

RadiusLeft Property

In conjunction with the RadiusTop property defines the extent of the brush.

Equal values for RadiusLeft and RadiusTop will ensure that the brush is round, center and origin values nothwithstanding. By using differing values, the brush will appear squashed.

RadiusLeft is a percentage.

Both RadiusLeft and RadiusTop have a default of 50.

RadiusTop Property

In conjunction with the RadiusLeft property defines the extent of the brush.

Equal values for RadiusLeft and RadiusTop will ensure that the brush is round, center and origin values nothwithstanding. By using differing values, the brush will appear squashed.

RadiusLeft is a percentage.

Both RadiusLeft and RadiusTop have a default of 50.

Solid Brush

A Solid Brush is effectively a Linear Brush (#Prim_VS.LinearBrush) with a single color. This avoids the need to create the colors (#Prim_VS.BrushColors) and color (#Prim_VS.BrushColor) and allows for a simple definition on one statement.

Color Property

Denotes the color of the brush.

Color can be any valid RGB value e.g. 255:0:0 for red, or any of a series of predefined named colors such as red, blue, yellow or transparent.

Opacity Property

Opacity denotes the appearance of a gradient in terms of its interaction with its background.

The default is 100; entirely opaque. Nothing behind the gradient will be visible at all.

As the value decreases, more and more of the background will become visible through the control. When the value is 0 the control will be entirely transparent and only the background will be visible.

When specified on a control, opacity affects the whole of the control, both foreground and background. When specified on a brush, opacity can be used to target either the foreground or the background, allowing for a semi-transparent background, but still fully opaque text.

Image Brush

An Image Brush is used to apply an image to a control rather than colors. A typical use for this would be as watermark or background image for an application.

Alignment Property

Alignment determines where the image is located when it does not fill the space available. Conversely, if the image is too large for the space, Alignment determines which portion of the image is visible.

Possible values are – TopLeft, TopCenter, TopRight, CenterLeft, Center, CenterRight, BottomLeft, BottomCenter & BottomRight

Height Property

Determines the height of the image in pixels or as a percentage dependent on the Units property. When tiled, this denotes the size of each tile.

Image Property

Reference to an instance of a basic LANSA graphic component (#PRIM_FLBX).

This can an enrolled bitmap (#Prim_Bmp) or icon (#Prim_Icon), or a bitmap that has been created at runtime.

Left Property

Specifies how far from the left hand edge of the control the image is shown, either as a specific number of pixels or as a percentage of the control's width as specified in the Units property.

Opacity Property

Opacity denotes the appearance of a gradient in terms of its interaction with its background.

The default is 100; entirely opaque. Nothing behind the image will be visible at all.

As the value decreases, more and more of the background will become visible through the control. When the value is 0 the control will be entirely transparent and only the background will be visible.

When specified on a control, opacity affects the whole of the control, both foreground and background. When specified on a brush, opacity can be used to target either the foreground or the background, allowing for a semi-transparent background, but still fully opaque text.

Sizing Property

Describes how the image is stretched or squashed to fit in to the available container, be it a control or a tiled portion. The default is None.

None

The image is not resized in any way

Best Fit

The image is resized to fit in the available space while still maintaining its aspect ratio.

Fit Both

The image is resized to precisely fit the horizontal and vertical space. The aspect ratio is not maintained. This may result in the image appearing stretched.

Cropped

The image is resized to fit the available space while still maintaining its aspect ratio. Once either the horizontal or vertical fits within the control, not further resizing is done. This may result in some of the image not being visible.

Tile Property

Describes how the image is repeated within the control.

The default is None. All other enumeration values will result in the image being repeated both vertically and horizontally.

None

The image is not tiled in any way

Tile

The image is image repeated both vertically and horizontally.

MirrorHorizontal

The image is tiled and reflected along the vertical axis.

MirrorVertical

The image is tiled and reflected along the horizontal axis.

MirrorBoth

The image is tiled and reflected along both the horizontal and vertical axes.

Top Property

Specifies how far from the upper edge of the control the image is shown, either as a specific number of pixels or as a percentage of the control's width as specified in the Units property.

Units Property

Denotes how the size and position of the image is measured. The default is Percentage

Percentage

All height, width, left and top values are measured as a percentage of the control.

Pixels

All height, width, left and top values are measured as a specific number of pixels.

Width Property

Determines the width of the image in pixels or as a percentage dependent on the Units property. When tiled, this denotes the size of each tile.

Visual Brush

An Image Brush is used to apply an image to a control rather than colors. However, unlike Image Brush, which uses a static image, Visual Brush uses another control.

A typical use for this would be as the image when dragging and dropping.

Control Property

Reference to an instance of a Control (#Prim_CTRL).

As the image seen is the actual control itself, the control must be realized.

The image is not a snapshot or copy of the source image, but rather a reference to it. This means that the image will respond to changes made to the control.

Effect

DirectX Only

Effects are used to augment the features specified on the style. There are two types of effect: Blur (#Prim_VS.Blur) or DropShadow (#Prim_VS.DropShadow)

Blur Effect

The blur effect, as the name suggests, is used to blur a control.
KernelType Property

There are two types of blur.

Gaussian

This results in a smoother blur with a higher quality. However, this takes more processing to achieve.

Box

This results in simpler blur with a lower quality that takes less processing.

Radius Property

Determines the blurriness of the blur.

DropShadow Effect

Drop shadows can be used to make controls appear as though floating above their parent control. They can also be used to create a glow like effect, typically behind text, as seen in the title bar of an Aero theme Windows form.

Like scale and rotation, Dropshadow is a visual effect and does not alter the fundamental size of the control.

Controls with a DropShadow are clipped to ensure they fit within the bounds of their parent control.

BlurRadius Property

Denotes the amount of blur to be applied to the shadow. The closer the value is to zero, the less blurred the shadow.

Color Property

Denotes the color of the drop shadow. The default is silver.

Direction Property

In conjunction with the ShadowDepth property, denotes the position of the drop shadow in relation to the source control.

Dropshadow is measured in degrees.

A value of zero is equivalent to having the light source casting the shadow at 9 o'clock.

Opacity Property

Opacity denotes the appearance of a shadow in terms of its interaction with its background.

The default is 100; entirely opaque. Nothing behind the shadow will be visible at all.

As the value decreases, more and more of the background will become visible through the shadow. When the value is 0 the shadow will be entirely transparent and only the background will be visible.

ShadowDepth Property

In conjunction with the Direction property, denotes the position of the drop shadow in relation to the source control.

A value of zero result in the shadow being directly behind the control.

User Designed Controls

Tile (Prim_Tile)

DirectX Only

Tile is a member of a group of list related primitive controls referred to as User Designed Controls.

Tile organises individual design panels in to a grid that is the equivalent of a flow layout manager.

UDCs can be manipulated by the use of the typical LANSA list commands ADD_ENTRY, UPD_ENTRY etc. When entries are added to the list, an instance of the design is made, fields can be passed in, and a corresponding list item is made. UDCs control the position of the item within the list, manage selection, focus, expand/collapse etc., and communicate with the individual item designs through a series of predefined methods published on an interface specific to the type of UDC. For Prim_Tile this is #Prim_Tile.iTileDesign All UDCs use a parameterized type to define the class of the design to be

All UDCs use a parameterized type to define the class of the design to be created each time an entry is added. This is specified on the DEFINE_COM as below

```
Define_Com Class(#Prim_Tile<#MyTileDesign>) Name(#Tile)
```

Because of the overhead of making a reusable part instance compared to a simple item for a tree view (#Prim_TRVW) or list view (#Prim_LTVW), UDCs are not designed to be used with many thousand items. For high volume scenarios, other techniques are recommended.

Items Property

Collection of all the items currently in the Tile.

SelectionStyle Property

Defines whether Tile allows one or more than one item to be selected.

Single

Only one item can be selected.

Mulitple

Multiple items can be selected.

Tile Event Item

DirectX Only

Reference to an instance of a Tile Item (#Prim_Tile.iTileItem).

The item supplied on the event is a reference to the item currently being processed.

Add Method

Rather than using the typical LANSA list commands, the Add method allows different classes of design to be added.

When the Add method is used no field values can be passed to the design using the *ListFields feature. The user is in control of the process and needs to pass any data in to the new design instance programmatically.

Having the ability to add different types of design makes dealing with complex UI requirements far simpler.

Result Parameter

Reference to the Item (#Prim_Tile.TileItem) created by the Add method.

DesignType Parameter

Class of the design instance to be created.

The class specified must inherit from the class specified as the parameterized type on the Tile DEFINE_COM.

DeleteAll Method

As the name suggests, this method deletes all items in the list. This is equivalent to using the Clr_list command.

FindItem Method

FindItem calls the OnFind method on the Design Interface for each item. Each Tile item is called in sequence, receiving the value to be tested and returning a boolean result. Once a positive result is returned, the Find will end. It can be started again from the item that returns the result.

Mthroutine Name(Search)

Define_Com Class(#Prim_Tile.TileItem) Name(#FoundItem) Reference(*dynamic) Begin_Loop #FoundItem <= #Tree.FindItem(#Search #FoundItem) Leave If(#FoundItem *Is *null)

End_Loop Endroutine

Result Parameter

Reference to the first item (#Prim_Tile.TileItem) to return a positive result.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

Item Parameter

Reference to the Item (#Prim_Tile.TileItem) after which to start the search. If no StartItem is specified, the search will start at the beginning.

iTileDesign Interface

The iTileDesign must be implemented by any reusable part that is to be used as a Design for a Tile control (#Prim_Tile). The interface contains a series of methods through which the Tile can communicate with the design instance to inform it of changes to its state e.g. Selection, Focus etc.

The source below shows the Begin_com for reusable part implementing the appropriate design for Prim_tile as well as the *Listfields feature which defines the fields to be received by the design when it is ADD_ENTRY is used to create the item.

Begin_Com Role(*EXTENDS #Prim_Panl3 *implements #Prim_Tile.iTileDesign *ListFields #ListFields)

* Fields received as on Add_entry

Group_By Name(#ListFields) Fields(#Empno #Surname #Givename #Deptment #Section #Deptdesc #Secdesc)

OnAdd Method

OnAdd is executed when the design instance is first added to the list via either ADD_ENTRY or use of the ADD Method.

Any fields specified in the *ListFields feature of Begin_Com Role parameter will have been populated prior to the execution of this method when using ADD_ENTRY.

Tile Item

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) created when the design was added.

OnDelete Method

OnDelete is executed when corresponding item is in the process of being deleted.

The design instance will be destroyed as part of the delete, assuming no other references to the design exist.

Tile Item

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) being deleted.

OnDisplaying Method

OnDisplaying is executed whenever the Design is about to be in the visible portion of the User Design Control.

This allows the user to delay the processing of the Design until it is necessary rather than when it is created. This is useful for situations where the UDC has many items or the processing is relatively long running.

Tile Item

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) for the design about to be displayed.

OnFind Method

OnFind is executed when the Find method is used on the parent user designed control.

Each item is called in sequence, receiving the value to be tested and returning a boolean result. Once a positive result is returned, the Find will end. It can be started again from the item that returns the result.

Mthroutine Name(OnFind) Options(*Redefine) #Result := #Surname.Contains(#Key) Endroutine

Result Parameter

Set the Result to indicate whether the Find was successful.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

TileItem Parameter

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) for the design.

OnItemGotFocus Method

Executed when the design instance becomes the focus instance.

Similar to a list or tree, an item becomes focus when the user clicks or positions the cursor within the bounds of the design.

There is no automatic visual feedback when the item gets or loses focus. The user is responsible for adding/removing styles.

TileItem Parameter

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) for the design.

OnItemLostFocus Method

Executed when the design instance is about to lose focus.

There is no automatic visual feedback when the item gets or loses focus. The user is responsible for adding/removing styles.

TileItem Parameter

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) for the design.
OnItemGotSelection Method

Executed when the design instance becomes selected.

Similar to a list or tree, an item becomes selected when the user clicks or positions the cursor within the bounds of the design.

There is no automatic visual feedback when the item gets or loses selection. The user is responsible for adding/removing styles.

TileItem Parameter

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) for the design.

OnItemLostSelection Method

Executed when the design instance is about to lose selection.

There is no automatic visual feedback when the item gets or loses selection.

The user is responsible for adding/removing styles.

TileItem Parameter

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) for the design.

OnUpdate Method

OnUpdate is executed when UPD_ENTRY is used to update the corresponding item.

Tile Item

Reference to the corresponding Tile Item (#Prim_Tile.TileItem) being updated.

Tile Property

Reference to the parent Tile (#Prim_Tile) control

Tile Item (Prim_Tile.TileItem)

Tile Items are created whenever an entry is added to the Tile (Prim_Tile) e.g. ADD_ENTRY or the Tile Add method.

For every item that is added, a corresponding user defined Design instance is created that provides the visual portion of the control.

Design Property

Reference to a Tile Design (#Prim_Tile.iTileDesign)

Provides access to the corresponding Design instance created when the entry was added to the Tile.

Tile Property

Reference to the parent Tile (#Prim_Tile) control

Carousel (Prim_Caro)

DirectX Only

Carousel is a member of a group of list related primitive controls referred to as User Designed Controls.

Carousel organises individual design panels, typically images, in to linear or elliptical sequence with only one central design panel being fully visible

UDCs can be manipulated by the use of the typical LANSA list commands ADD_ENTRY, UPD_ENTRY etc. When entries are added to the list, an instance of the design is made, fields can be passed in, and a corresponding list item is made. UDCs control the position of the item within the list, manage selection, focus, expand/collapse etc., and communicate with the individual item designs through a series of predefined methods published on an interface specific to the type of UDC. For Prim_Caro this is #Prim_Caro.iCarouselDesign

All UDCs use a parameterized type to define the class of the design to be created each time an entry is added. This is specified on the DEFINE_COM as below

Define_Com Class(#Prim_Caro<#MyCarouselDesign>) Name(#Carousel)

Because of the overhead of making a reusable part instance compared to a simple item for a tree view (#Prim_TRVW) or list view (#Prim_LTVW), UDCs are not designed to be used with many thousand items. For high volume scenarios, other techniques are recommended.

CarouselStyle Property

A carousel can be viewed in two ways

Linear

Items will be organised across the screen, left to right and equally spaced. The control will only show as many item as can fit in the control.

Elliptical

Designs will be organised in an elliptical fashion.

The ellipse is sized to fit on the screen and all items will be visible.

Items Property

Collection of all the items currently in the Carousel.

NavigationStyle Property

Buttons

Show the default navigation buttons.

None

Hide the default navigation buttons.

Carousel Event Item

DirectX Only

Reference to an instance of a Carousel Item (#Prim_Caro.CarouselItem). The item supplied on the event is a reference to the item currently being processed.

Add Method

Rather than using the typical LANSA list commands, the Add method allows different classes of design to be added.

When the Add method is used no field values can be passed to the design using the *ListFields feature. The user is in control of the process and needs to pass any data in to the new design instance programmatically.

Having the ability to add different types of design makes dealing with complex UI requirements far simpler.

Result Parameter

Reference to the Item (#Prim_Caro.CarouselItem) created by the Add method.

DesignType Parameter

Class of the design instance to be created.

The class specified must inherit from the class specified as the parameterized type on the Carousel DEFINE_COM.

DeleteAll Method

As the name suggests, this method deletes all items in the list. This is equivalent to using the Clr_list command.

FindItem Method

FindItem calls the OnFind method on the Design Interface for each item.

Each item is called in sequence, receiving the value to be tested and returning a boolean result. Once a positive result is returned, the Find will end. It can be started again from the item that returns the result.

Mthroutine Name(Search) Define_Com Class(#Prim_Caro.CarouselItem) Name(#FoundItem) Reference(*dynamic) Begin_Loop #FoundItem <= #Tree.FindItem(#Search #FoundItem) Leave If(#FoundItem *Is *null) End_Loop Endroutine

Result Parameter

Reference to the first item (#Prim_Caro.CarouselItem) to return a positive result.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

Item Parameter

Reference to the Item (#Prim_Caro.CarouselItem) after which to start the search.

If no StartItem is specified, the search will start at the beginning.

FirstItem Method

Set the first item in the Carousel to be the focus item.

Animate Parameter

True

Animate the move showing the some of the items between the current item and the first

False

Jump to the first item without showing any other items

LastItem Method

Set the last item in the Carousel to be the focus item.

Animate Parameter

True

Animate the move showing the some of the items between the current item and the last

False

Jump to the last item without showing any other items

MoveToItem Method

Move to the item specified in the Position parameter.

Position Parameter

Specifies the item to become the active item.

Animate Parameter

True

Animate the move showing the some of the items between the current item and the target item

False

Jump to the target item without showing any other items

NextItem Method

Set the next item in the Carousel to be the focus item.

Animate Parameter

True

Animate the move to the next item

False

Jump to the next item without animating

PrevItem Method

Set the previous item in the Carousel to be the focus item.

Animate Parameter

True

Animate the move to the previous item

False

Jump to the previous item without animating

iCarouselDesign Interface

The iCarouselDesign must be implemented by any reusable part that is to be used as a Design for a Carousel control (#Prim_Caro). The interface contains a series of methods through which the Carousel can communicate with the design instance to inform it of changes to its state e.g. Selection, Focus etc.

The source below shows the Begin_com for reusable part implementing the appropriate design for Prim_Caro as well as the *Listfields feature which defines the fields to be received by the design when it is ADD_ENTRY is used to create the item.

Begin_Com Role(*EXTENDS #Prim_Panl *implements #Prim_Caro.iCarouselDesign *ListFields #ListFields)

* Fields received as on Add_entry

Group_By Name(#ListFields) Fields(#Empno #Surname #Givename #Deptment #Section #Deptdesc #Secdesc)

CarouselItem Parameter

Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) created when the design was added.
Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) being deleted.

Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) for the design about to be displayed.

Result Parameter

Set the Result to indicate whether the Find was successful.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) for the design.

Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) for the design.

Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) for the design.

Reference to the corresponding Carousel Item (#Prim_Caro.CarouselItem) being updated.

Carousel Items (Prim_Carousel.iCarouselItems)

Collection of all items currently in the Carousel (#Prim_Caro)

Carousel Property

Reference to the parent Carousel (#Prim_Caro) control

Carousel Item (Prim_Carousel.iCarouselItem)

Carousel Items are created whenever an entry is added to the Carousel (Prim_Caro) e.g. ADD_ENTRY or the Carousel Add method.

For every item that is added, a corresponding user defined Design instance is created that provides the visual portion of the control.

Book (Prim_Book)

DirectX Only

Book is a member of a group of list related primitive controls referred to as User Designed Controls.

Book organises individual design panels as though they were pages of a book. UDCs can be manipulated by the use of the typical LANSA list commands ADD_ENTRY, UPD_ENTRY etc. When entries are added to the list, an instance of the design is made, fields can be passed in, and a corresponding list item is made. UDCs control the position of the item within the list, manage selection, focus, expand/collapse etc., and communicate with the individual item designs through a series of predefined methods published on an interface specific to the type of UDC. For Prim_Book this is #Prim_Book.iBookDesign All UDCs use a parameterized type to define the class of the design to be created each time an entry is added. This is specified on the DEFINE_COM as below

```
Define_Com Class(#Prim_Book<#MyBookDesign>) Name(#Book)
```

Because of the overhead of making a reusable part instance compared to a simple item for a tree view (#Prim_TRVW) or list view (#Prim_LTVW), UDCs are not designed to be used with many thousand items. For high volume scenarios, other techniques are recommended.

Items Property

Collection of all the items currently in the Book.

NavigationStyle Property

Buttons

Show the default navigation buttons.

None

Hide the default navigation buttons.

Book Event Item

DirectX Only

Reference to an instance of a Book Item (#Prim_Book.BookItem).

The item supplied on the event is a reference to the item currently being processed.

Add Method

Rather than using the typical LANSA list commands, the Add method allows different classes of design to be added.

When the Add method is used no field values can be passed to the design using the *ListFields feature. The user is in control of the process and needs to pass any data in to the new design instance programmatically.

Having the ability to add different types of design makes dealing with complex UI requirements far simpler.

Result Parameter

Reference to the Item (#Prim_Book.BookItem) created by the Add method.

DesignType Parameter

Class of the design instance to be created.

The class specified must inherit from the class specified as the parameterized type on the Book DEFINE_COM.

DeleteAll Method

As the name suggests, this method deletes all items in the list. This is equivalent to using the Clr_list command.

FindItem Method

FindItem calls the OnFind method on the Design Interface for each item.

Each item is called in sequence, receiving the value to be tested and returning a boolean result. Once a positive result is returned, the Find will end. It can be started again from the item that returns the result.

Mthroutine Name(Search) Define_Com Class(#Prim_Book.BookItem) Name(#FoundItem) Reference(*dynamic) Begin_Loop #FoundItem <= #Tree.FindItem(#Search #FoundItem) Leave If(#FoundItem *Is *null) End_Loop Endroutine

Result Parameter

Reference to the first item (#Prim_Book.BookItem) to return a positive result.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

Item Parameter

Reference to the Item (#Prim_Book.BookItem) after which to start the search. If no StartItem is specified, the search will start at the beginning.

FirstItem Method

Set the first item in the Book to be the focus item.

Animate Parameter

True

Animate the move showing the some of the items between the current item and the first

False

Jump to the first item without showing any other items

LastItem Method

Set the last item in the Book to be the focus item.

Animate Parameter

True

Animate the move showing the some of the items between the current item and the last

False

Jump to the last item without showing any other items

MoveToItem Method

Move to the item specified in the Position parameter.

Position Parameter

Specifies the item to become the active item.

Animate Parameter

True

Animate the move showing the some of the items between the current item and the target item

False

Jump to the target item without showing any other items

NextItem Method

Set the next item in the Book to be the focus item.

Animate Parameter

True

Animate the move to the next item

False

Jump to the next item without animating

PrevItem Method

Set the previous item in the Book to be the focus item.

Animate Parameter

True

Animate the move to the previous item

False

Jump to the previous item without animating

iBookDesign Interface

The iBookDesign must be implemented by any reusable part that is to be used as a Design for a Book control (#Prim_Book). The interface contains a series of methods through which the Book can communicate with the design instance to inform it of changes to its state e.g. Selection, Focus etc.

The source below shows the Begin_com for reusable part implementing the appropriate design for Prim_Book as well as the *Listfields feature which defines the fields to be received by the design when it is ADD_ENTRY is used to create the item.

Begin_Com Role(*EXTENDS #Prim_Panl3 *implements #Prim_Book.iBookDesign *ListFields #ListFields)

* Fields received as on Add_entry

Group_By Name(#ListFields) Fields(#Empno #Surname #Givename #Deptment #Section #Deptdesc #Secdesc)

BooklItem Parameter

Reference to the corresponding Book Item (#Prim_Book.BookItem) created when the design was added.
Reference to the corresponding Book Item (#Prim_Book.BookItem) being deleted.

Reference to the corresponding Book Item (#Prim_Book.BookItem) for the design about to be displayed.

Result Parameter

Set the Result to indicate whether the Find was successful.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

Book Property

Reference to the parent Book (#Prim_Book) control

Book Item (Prim_Book.BookItem)

Book Items are created whenever an entry is added to the Book (Prim_Book) e.g. ADD_ENTRY or the Book Add method.

For every item that is added, a corresponding user defined Design instance is created that provides the visual portion of the control.

Tree (Prim_Tree)

DirectX Only

Tree is a member of a group of list related primitive controls referred to as User Designed Controls.

Tree organises individual design panels in to a hierarchical structure very similar to that of tree view (#Prim_trvw).

UDCs can be manipulated by the use of the typical LANSA list commands ADD_ENTRY, UPD_ENTRY etc. When entries are added to the list, an instance of the design is made, fields can be passed in, and a corresponding list item is made. UDCs control the position of the item within the list, manage selection, focus, expand/collapse etc., and communicate with the individual item designs through a series of predefined methods published on an interface specific to the type of UDC. For Prim_Tree this is #Prim_Tree.iTreeDesign

All UDCs use a parameterized type to define the class of the design to be created each time an entry is added. This is specified on the DEFINE_COM as below

Define_Com Class(#Prim_Tre<#MyTreeDesign>) Name(#Tree)

Because of the overhead of making a reusable part instance compared to a simple item for a tree view (#Prim_TRVW) or list view (#Prim_LTVW), UDCs are not designed to be used with many thousand items. For high volume scenarios, other techniques are recommended.

Items Property

Collection of all the items currently in the Tree.

Selection Style Property

All

Any item in the tree can be selected regardless of its position

Single

Only one item can be selected at any given moment

SameParent

Mulitple items can be selected provided they are all parented to the same item

SameLevel

Mulitple items can be selected provided they are all parented to the items with the same parent

Tree Event Item

DirectX Only

Reference to an instance of a Tree Item (#Prim_Tree.TreeItem).

The item supplied on the event is a reference to the item currently being processed.

ItemCollapsed Event

An item in the tree has just collapsed. All child nodes will no longer be visible to the user.

This corresponds to the OnItemCollapsed Method of the Tree design interface (#Prim_Tree.iTreeDesign)

ItemExpanding Event

An item in the tree is expanding. All child nodes will be shown immediately below the expanding item.

This corresponds to the OnItemExpanding Method of the Tree design interface (#Prim_Tree.iTreeDesign)

ItemGotParent Event

The ParentItem property of a tree item has been assigned to either another item in the tree or null.

This corresponds to the OnItemGotParent Method of the Tree design interface (#Prim_Tree.iTreeDesign)

Add Method

Rather than using the typical LANSA list commands, the Add method allows different classes of design to be added.

When the Add method is used no field values can be passed to the design using the *ListFields feature. The user is in control of the process and needs to pass any data in to the new design instance programmatically.

Having the ability to add different types of design makes dealing with complex UI requirements far simpler.

Result Parameter

Reference to the Item (#Prim_Tree.TreeItem) created by the Add method.

DesignType Parameter

Class of the design instance to be created.

The class specified must inherit from the class specified as the parameterized type on the Tree DEFINE_COM.

DeleteAll Method

As the name suggests, this method deletes all items in the list. This is equivalent to using the Clr_list command.

FindItem Method

FindItem calls the OnFind method on the Design Interface for each item.

Each Tree item is called in sequence, receiving the value to be tested and returning a boolean result. Once a positive result is returned, the Find will end. It can be started again from the item that returns the result.

Mthroutine Name(Search) Define_Com Class(#Prim_Tree.TreeItem) Name(#FoundItem) Reference(*dynamic) Begin_Loop #FoundItem <= #Tree.FindItem(#Search #FoundItem) Leave If(#FoundItem *Is *null) End_Loop Endroutine

Result Parameter

Reference to the first item (#Prim_Tree.TreeItem) to return a positive result.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

Item Parameter

Reference to the Item (#Prim_Tree.TreeItem) after which to start the search. If no StartItem is specified, the search will start at the beginning.

iTreeDesign Interface

The iTreeDesign must be implemented by any reusable part that is to be used as a Design for a Tree control (#Prim_Tree). The interface contains a series of methods through which the Tree can communicate with the design instance to inform it of changes to its state e.g. Selection, Focus etc.

The source below shows the Begin_com for reusable part implementing the appropriate design for Prim_tree as well as the *Listfields feature which defines the fields to be received by the design when it is ADD_ENTRY is used to create the item.

Begin_Com Role(*EXTENDS #Prim_Panl3 *implements #Prim_Tree.iTreeDesign *ListFields #ListFields)

* Fields received as on Add_entry

Group_By Name(#ListFields) Fields(#Empno #Surname #Givename #Deptment #Section #Deptdesc #Secdesc)

Tree Item Parameter

Reference to the corresponding Tree Item (#Prim_Tree.TreeItem) created when the design was added.

Tree Item Parameter

Reference to the corresponding Tree Item (#Prim_Tree.TreeItem) being deleted.

Tree Item Parameter

Reference to the corresponding Tree Item (#Prim_Tree.TreeItem) for the design about to be displayed.

Result Parameter

Set the Result to indicate whether the Find was successful.

Key Parameter

Value or object to be searched for.

Key is a variant allowing it to carry any type of value or object.

TreeItem Parameter

Reference to the corresponding Tree Item (#Prim_Tree.TreeItem) for the design.

OnItemCollapsed Method

Executed when the corresponding item collapsed through the use of the Collapse method or by setting the Expanded property to false.

This method allows the user to change the design appearance to indicate that the item has collapsed. E.g. Show an appropriate collapsed image.

TreeItem Parameter

Reference to the corresponding Tree Item (#Prim_Tree.TreeItem) for the design.

OnItemExpanding Method

Executed when the corresponding item is expanded through the use of the Expand method or by setting the Expanded property to true.

This method allows the user to change the design appearance to indicate that the item has collapsed. E.g. Show an appropriate expanded image.

TreeItem Parameter

Reference to the corresponding Tree Item (#Prim_Tree.TreeItem) for the design.
OnItemGotParent Method

Executed when the corresponding item's Parent is changed by setting the ParentItem Property.Expand

This method allows the user to change the design appearance to indicate that the item has changed parent e.g. increment the left margin to increase the indent.

Tree Property

Reference to the parent Tree (#Prim_Tree) control

Tree Item (Prim_Tree.TreeItem)

Tree Items are created whenever an entry is added to the Tree (#Prim_Tree) e.g. ADD_ENTRY or the Tree Add method.

For every item that is added, a corresponding user defined Design instance is created that provides the visual portion of the control.

Design Property

Reference to a Tree Design (#Prim_Tree.iTreeDesign)

Provides access to the corresponding Design instance created when the entry was added to the Tree.

Expanded Property

The item is expanded. All child items will be visible.

FocusedStyle Property

Reference to a Style (#Prim_Vs.Style) to be applied to the design when the item becomes the focus item.

The Style will be removed as soon as the item loses focus.

Mthroutine Name(OnAdd) Option(*Redefine) #TreeItem.FocusStyle <= # MyStyles<Focus> Endroutine

FocusedStyles Property

Collection of Style (#Prim_Vs.Style) to be applied to the design when the item becomes the focus item.

The Styles will be removed as soon as the item loses focus.

MarginBottom Property

Defines the size in pixels of the margin to be applied to the bottom of the item.

MarginLeft Property

Defines the size in pixels of the margin to be applied to the left of the item.

MarginRight Property

Defines the size in pixels of the margin to be applied to the right of the item.

MarginTop Property

Defines the size in pixels of the margin to be applied to the top of the item.

MouseOverStyle Property

DirectX Only

Denotes the Style (#Prim_vs.Style) to be applied to a control in the event of the Mouse entering the physical bounds of the control. The style is removed when the mouse leaves the control.

MouseOverStyle negates the need to code many MouseEnter and corresponding MouseLeave events and allows for simple declaration instead.

MouseOverStyles Property

DirectX Only

Collection of Styles (#Prim_vs.Style) to be applied to a control in the event of the Mouse entering the physical bounds of the control. The styles are removed when the mouse leaves the control.

MouseOverStyles allows for more complex programmatic appearance changes to be coded. Rather than relying on the declarative single MouseOverStyle, the developer is free to add as many style layers as required.

ParentItem Property

Reference to a TreeItem (#Prim_Tree.TreeItem)

By setting the ParentItem property items within the tree can be formed in to complex hierarchies.

TreeItems cannot be parented to items in different Trees.

SelectedStyle Property

Reference to a Style (#Prim_Vs.Style) to be applied to the design when the item becomes selected.

The Style will be removed as soon as the item loses selection.

Mthroutine Name(OnAdd) Option(*Redefine) #TreeItem.SelectedStyle <= # MyStyles<Focus> Endroutine

SelectedStyles Property

Collection of Style (#Prim_Vs.Style) to be applied to the design when the item becomes selected.

The Styles will be removed as soon as the item loses selection.

Style Property

Denotes the Style (#Prim_vs.Style) to be applied to an item. When the Style property is set, all Styles previously applied to the item are removed.

Styles Property

Collection of Styles (#Prim_vs.Style) to be applied to a control.

Styles allows for more complex programmatic appearance changes to be coded. Rather than relying on the declarative single Style property, the developer is free to add as many Style layers as required.

Tree Property

Reference to the parent Tree (#Prim_Tree) control

System Application Controls

Focus Control Changed Event

FocusControlChanged is fired whenever focus moves to a different control. This is equivalent to the GotFocus event found on all controls (#Prim_CTRL)

Control Parameter

Reference to the control (#Prim_CTRL) that is focus.

Appearance Property

Reference to a Prim_Vs.Appearance instance.

The Appearance property provides a central location for default styles for each control to be applied, either at application startup or at runtime.

Calendar Property

Reference to a Style (#Prim_VS) to be applied to all calendar (#Prim_MTCL) instances.

CheckBox Property

Reference to a Style (#Prim_VS) to be applied to all check box (#Prim_CKBX) instances.

ComboBox Property

Reference to a Style (#Prim_VS) to be applied to all combo box (#Prim_CMBX) instances.

DateTime Property

Reference to a Style (#Prim_VS) to be applied to all DateTime (#Prim_CMBX) instances including date time fields.

Edit Property

Reference to a Style (#Prim_VS) to be applied to all edit (#Prim_EDIT) instances. This includes edits when used as part of an entry field (#Prim_EVEF)

Grid Property

Reference to a Style (#Prim_VS) to be applied to all grid (#Prim_GRID) instances.

GridFocused Property

Reference to a Style (#Prim_VS) to be applied to all grid item (#Prim_GDIT) instances when they become focus and the grid is the focus control.

GridFocusedInactive Property

Reference to a Style (#Prim_VS) to be applied to all grid item (#Prim_GDIT) instances when they are the focus item but the grid is not the focus control.

GridSelectedInactive Property

Reference to a Style (#Prim_VS) to be applied to all grid item (#Prim_GDIT) instances when they are selected, but the grid is not the focus control.
GridMouseOver Property

Reference to a Style (#Prim_VS) to be applied to all grid item (#Prim_GDIT) instances when the mouse enters their bounds.

GridSelected Property

Reference to a Style (#Prim_VS) to be applied to all grid item (#Prim_GDIT) instances when they become selected and the grid is the focus control.

GroupBox Property

Reference to a Style (#Prim_VS) to be applied to all group box (#Prim_GPBX) instances.

Image Property

Reference to a Style (#Prim_VS) to be applied to all image (#Prim_IMGE) instances.

Label Property

Reference to a Style (#Prim_VS) to be applied to all label (#Prim_LABL) instances.

List Box Property

Reference to a Style (#Prim_VS) to be applied to all list box (#Prim_LTBX) instances.

ListBoxFocused Property

Reference to a Style (#Prim_VS) to be applied to all list box item (#Prim_LBIT) instances when they become focus and the grid is the focus control.

ListboxFocusedInactive Property

Reference to a Style (#Prim_VS) to be applied to all listbox item (#Prim_LBIT) instances when they are the focus item but the listbox is not the focus control.

ListboxSelectedInactive Property

Reference to a Style (#Prim_VS) to be applied to all listbox item (#Prim_LBIT) instances when they are selected, but the listbox is not the focus control.

ListboxSelectedInactive Property

Reference to a Style (#Prim_VS) to be applied to all listview item (#Prim_LVIT) instances when they are selected, but the listview is not the focus control.

ListBoxMouseOver Property

Reference to a Style (#Prim_VS) to be applied to all list box item (#Prim_LBIT) instances when the mouse enters their bounds.

ListBoxSelected Property

Reference to a Style (#Prim_VS) to be applied to all list box item (#Prim_LBIT) instances when they become selected and the grid is the focus control.

List View Property

Reference to a Style (#Prim_VS) to be applied to all list view (#Prim_LTVW) instances.

ListViewFocused Property

Reference to a Style (#Prim_VS) to be applied to all list view item (#Prim_LVIT) instances when they become focus and the list view is the focus control.

ListviewFocusedInactive Property

Reference to a Style (#Prim_VS) to be applied to all listview item (#Prim_LVIT) instances when they are the focus item but the listview is not the focus control.

ListboxSelectedInactive Property

Reference to a Style (#Prim_VS) to be applied to all listview item (#Prim_LVIT) instances when they are selected, but the listview is not the focus control.

ListViewMouseOver Property

Reference to a Style (#Prim_VS) to be applied to all list view item (#Prim_LVIT) instances when the mouse enters their bounds.

ListViewSelected Property

Reference to a Style (#Prim_VS) to be applied to all list view item (#Prim_LVIT) instances when they become selected and the list view is the focus control.

Memo Property

Reference to a Style (#Prim_VS) to be applied to all memo (#Prim_MEMO) instances.

Panel Property

Reference to a Style (#Prim_VS) to be applied to all panel (#Prim_PANL) instances.

Popup Property

Reference to a Style (#Prim_VS) to be applied to all popup panel (#Prim_PPNL) instances.

ProgressBar Property

Reference to a Style (#Prim_VS) to be applied to all progress bar (#Prim_PGBR) instances.

PushButton Property

Reference to a Style (#Prim_VS) to be applied to all push button (#Prim_PHBN) instances.

RadioButton Property

Reference to a Style (#Prim_VS) to be applied to all radio button (#Prim_RDBN) instances.

RadioButton Property

Reference to a Style (#Prim_VS) to be applied to all radio button (#Prim_RDBN) instances.

SpeedButton Property

Reference to a Style (#Prim_VS) to be applied to all speed button (#Prim_SPBN) instances.

SpinEdit Property

Reference to a Style (#Prim_VS) to be applied to all spin edit (#Prim_SPDT) instances.

Tab Property

Reference to a Style (#Prim_VS) to be applied to all tab (#Prim_TAB) instances.

TrackBar Property

Reference to a Style (#Prim_VS) to be applied to all spin edit (#Prim_TKBR) instances.

Tree Property

Reference to a Style (#Prim_VS) to be applied to all tree (#Prim_TRVW) instances.

TreeFocused Property

Reference to a Style (#Prim_VS) to be applied to all tree item (#Prim_TVIT) instances when they become focus and the tree is the focus control.

TreeviewFocusedInactive Property

Reference to a Style (#Prim_VS) to be applied to all treeview item (#Prim_TVIT) instances when they are the focus item but the treeview is not the focus control.

TreeboxSelectedInactive Property

Reference to a Style (#Prim_VS) to be applied to all treeview item (#Prim_TVIT) instances when they are selected, but the treeview is not the focus control.

TreeMouseOver Property

Reference to a Style (#Prim_VS) to be applied to all tree item (#Prim_ TVIT) instances when the mouse enters their bounds.

TreeSelected Property

Reference to a Style (#Prim_VS) to be applied to all tree item (#Prim_ TVIT) instances when they become selected and the tree view is the focus control.

Cursors Property

Collection of standard cursors (#Prim_CRSR) adopted from the Windows theme.

Cursors can be applied to controls so that the cursor changes when the mouse enters or leaves the bounds of a control. For most situations Visual LANSA will manage the cursor correctly without the developer needing to change it.

```
#Button.Cursor <= #sys_Appln.Cursors<Hand>
```
FocusControl Property

Reference to the control (#Prim_CTRL) that is focus.

GlassEnabled Property

Determines whether the application is running in an environment where a form may have a glass (#Prim_Form.Glass) appearance.

True

The form is configured to have a glass appearance and the application is currently capable of running with glass.

False

Either the form is not configured to have a glass appearance or the application is currently incapable of running with glass.

HelpHandlerProperty

Reference to an instance of a reusable a component that implements the HelpHandler Interface (#Prim_App.iHelpHandler)

By specifying a help handler, the developer can replace the default F1 help behavior.

Images Property

Collection of standard images (#Prim_BMP) adopted from the Windows theme. Images are useful when constructing user defined controls, particularly user trees (#Prim_TREE), so that applications can have a consistent look and feel.

#ExpandCollapse.image <= #sys_appln.Images<ExplorerCollapsed>

Operating System Property

Name of the operating system currently on which the application is executing.

PartitionShortCharLength Property

Provides access to the current partition short char. length based on the PartitionShortCharLevel property.

PartitionShortCharLevel Property

Provides access to the current partition short char. level property.

RenderStyle Property

Equivalent to the RNDR X_Run argument, RenderStyle provides access to the current rendering engine for the application

Win32

The application is using the Win32 rendering engine and no DirectX rendering is possible. This is equivalent to a RNDR value of W

Win32AndDirectX

The application is using the Win32 rendering engine but supports the use of DirectX on specific components. This is equivalent to a RNDR value of M

DirectX

The application is using the DirectX rendering engine but supports the use of Win32 for specified forms. This is equivalent to a RNDR value of X

Style Property

Reference to a Style (#Prin_VS.Style) to be applied to the whole application. The style specified will be adopted by all forms and reusable parts.

Styles Property

Collection of Styles (#Prin_VS.Style) to be applied to the whole application. The styles specified will be adopted by all forms and reusable parts.

TraceHandlerProperty

Reference to an instance of a reusable a component that implements the TraceHandler Interface (#Prim_App.iTraceHandler)

By specifying a trace handler, the developer can implement their own tracing mechanism.

Ribbon Controls

Ribbon

A ribbon is a command bar that organizes a program's features into a series of tabs or sheets at the top of a form. A ribbon combines the more traditional menu bar and toolbars into a single control concept.

Ribbon examples can be seen in the version 13 LANSA IDE and Microsoft Office 2007 and beyond.

Not all application environments are suited to a ribbon and you should consider carefully whether a ribbon is the best solution for your requirements.

Ribbons are always as wide as the component they are being used on and have a fixed height.

Minimized

True

The ribbon is shown minimized. Only the sheet captions will be visible

False

The entire ribbon is shown.

OpenPage

Reference to a ribbon sheet (#Prim_RBBN.Sheet) Provides access to the active ribbon sheet.

QuickAccessToolbarOnTop

True

The Quick Access Toolbar is displayed above the ribbon. In forms with a glass appearance, the toolbar will appear in the titlebar area.

False

The Quick Access Toolbar will be displayed immediately below the ribbon.

Access Key

Access Keys define shortcut commands (F3, Ctrl+S etc.,) and KeyTips for mnemonic access to be attached to specific controls on the ribbon.

Control

Reference to an instance of a Control (#Prim_CTRL) to which the keytip and shortcut will apply.

Application Menu

Application menu (#Prim_RBBN.ApplicationMenu) is shown on the left hand side of the popup displayed when the blue application button is clicked.

There are no specific design rules and as a result the application menu is just a plain panel. How it behaves and what is shown is entirely down to the developer.

Define_Com Class(#Prim_rbbn) Name(#Ribbon) Displayposition(1) Height(140) Left(8) Parent(#COM_OWNER) Tabposition(1) Top(8) Width(1033) Define_Com Class(#Prim_Rbbn.ApplicationMenu) Name(#ApplicationMenu) Caption('File') Displayposition(1) Height(80) Keytip('F') Left(30) Parent(#Ribbon) Tabposition(8) Top(0) Width(100) Define_Com Class(#Prim_Rbbn.ApplicationMenuContent) Name(#ApplicationMenuContent) Displayposition(5) Height(80) Parent(#Ribbon) Tabposition(7) Top(0) Define_Com Class(#Prim_Rbbn.ApplicationMenuFooter) Name(#ApplicationMenuFooter) Displayposition(4) Height(20) Left(30) Parent(#Ribbon) Tabposition(6) Top(0)

Caption

The Application Menu Caption will be shown in the blue application menu button, typically the "File" item.

As ribbon keyboard navigation is achieved through the use of AccessKeys (#Prim_RBBN.AccessKey) and KeyTips, there is no need to embed an & in the text.

Application Menu Content

Application menu content (#Prim_RBBN.ApplicationMenuContent) is shown on the right hand side of the popup displayed when the blue application button is clicked.

As the content may change entirely depending on what is active on the application content page it may be necessary to make many different content instances. As with the Application Menu itself, Content is a plain panel and the design and behaviour is ultimately down to the developer.

Define_Com Class(#Prim_rbbn) Name(#Ribbon) Displayposition(1) Height(140) Left(8) Parent(#COM_OWNER) Tabposition(1) Top(8) Width(1033)

Define_Com Class(#Prim_Rbbn.ApplicationMenu)

Name(#ApplicationMenu) Caption('File') Displayposition(1) Height(80) Keytip('F') Left(30) Parent(#Ribbon) Tabposition(8) Top(0) Width(100)

Define Com Class(#Prim Rbbn.ApplicationMenuContent)

Name(#ApplicationMenuContent) Displayposition(5) Height(80)

Parent(#Ribbon) Tabposition(7) Top(0)

Define_Com Class(#Prim_Rbbn.ApplicationMenuFooter)

Name(#ApplicationMenuFooter) Displayposition(4) Height(20) Left(30) Parent(#Ribbon) Tabposition(6) Top(0)

Application Menu Footer

The Application menu footer (#Prim_RBBN.ApplicationMenuFooter) is shown at the bottom of the popup displayed when the blue application button is clicked.

As with the Application Menu and Content, the Footer is a plain panel and the design and behaviour is ultimately down to the developer.

Define_Com Class(#Prim_rbbn) Name(#Ribbon) Displayposition(1) Height(140) Left(8) Parent(#COM_OWNER) Tabposition(1) Top(8) Width(1033) Define_Com Class(#Prim_Rbbn.ApplicationMenu) Name(#ApplicationMenu) Caption('File') Displayposition(1) Height(80) Keytip('F') Left(30) Parent(#Ribbon) Tabposition(8) Top(0) Width(100) Define_Com Class(#Prim_Rbbn.ApplicationMenuContent) Name(#ApplicationMenuContent) Displayposition(5) Height(80) Parent(#Ribbon) Tabposition(7) Top(0) Define_Com Class(#Prim_Rbbn.ApplicationMenuFooter) Name(#ApplicationMenuFooter) Displayposition(4) Height(20) Left(30) Parent(#Ribbon) Tabposition(6) Top(0)

Contextual Group

A Contextual Group can be used to collect sheets (#Prim_RBBN.Sheet) that belong to a similar group, or simply to display further information about a sheet in the title bar.

Microsoft Word uses contextual groups for features such as Tables. When a table is selected in the document an additional pair of ribbon sheets is shown.

Define_Com Class(#Prim_rbbn.ContextualGroup) Name(#ContextualGroup) Caption('Contextual Group') Parent(#Ribbon)

Define_Com Class(#Prim_rbbn.Sheet) Name(#Sheet1) Caption('Sheet1') Contextualgroup(#ContextualGroup) Displayposition(1) Height(91) Left(0) Parent(#Ribbon) Tabposition(3) Top(49) Width(1033)

Define_Com Class(#Prim_rbbn.Sheet) Name(#Sheet2) Caption('Sheet2') Contextualgroup(#ContextualGroup) Displayposition(2) Height(91) Left(0) Parent(#Ribbon) Tabposition(4) Top(49) Width(1033)

Caption

Text that will be displayed as a heading above the sheets that comprise the contextual group.

Visible

The visible property overrides the visible state for all sheets that are part of the group.

True

All sheets that are part of this contextual group will be visible, unless the sheet is Visible(False)

False

All sheets that are part of this contextual group will be hidden regardless.

Ribbon Group

A Group is a subset of a sheet. Most sheets will have several different groups that organize the commands on the ribbon into related subjects.

Controls can only be shown on a ribbon if they area parented to a Group

As with all parts of a ribbon, each group is a plain panel. Design is down the developer.

Define_Com Class(#Prim_rbbn) Name(#Ribbon) Displayposition(1) Height(140) Left(8) Parent(#COM_OWNER) Tabposition(1) Top(8) Width(1033) Define_Com Class(#Prim_rbbn.Sheet) Name(#Sheet1) Caption('Sheet1') Displayposition(2) Height(91) Left(0) Parent(#Ribbon) Tabposition(1) Top(49) Width(1033) Define_Com Class(#Prim_rbbn.Group) Name(#Sheet1Group1) Caption('Clipboard') Dialogbutton(True) Displayposition(1) Height(91) Left(12) Parent(#Sheet1) Tabposition(1) Top(0) Width(100) Define_Com Class(#Prim_rbbn.Group) Name(#Sheet1Group2) Caption('Styles') Displayposition(2) Height(91) Left(124) Parent(#Sheet1) Tabposition(2) Top(0) Width(100) Define_Com Class(#Prim_rbbn.Group) Name(#Sheet1Group3) Caption('Editing') Displayposition(3) Height(91) Left(236) Parent(#Sheet1) Tabposition(3) Top(0) Width(100)

DialogButton Click Event

Fired when the group Dialog Button is clicked.

DialogButton Property

True

Show the dialog button in the bottom right hard corner of the group

False

Hide the dialog button

Help Toolbar

The ribbon help toolbar is show in the top right hand corner of the ribbon.

As with all parts of a ribbon, the Help Toolbar is a plain panel. Design is down the developer.

Image Property

Reference to a base LANSA graphic (#Prim_FLBX) Denotes the 32x32 image to be displayed when the group is collapsed.

Quick Access Toolbar

The Quick Access Toolbar is shown on the ribbon on the left hand side. It typically displays a lines of small images used for commonly used commands such as Save. Whether it is shown above or below the Ribbon is governed by the Ribbon QuickAccessToolbarOnTop property.

As with all parts of a ribbon, the Quick Access Toolbar is a plain panel. Design is down the developer.

Ribbon Sheet

A Ribbon sheet is a container for Ribbon Groups (#Prim_RBBN.Group). Sheets define the tabbed pages that comprise the ribbon.

KeyTip

KeyTip is the string used to access a particular command on the ribbon when using the keyboard.

A Keytip is usually single character, typically the initial letter, but for more complex ribbons it may be preferable to use two characters.

Contextual Group

Reference to the Context Group (#Prim_RBBN.ContextualGroup).

Used to collect sheets in to set that can be displayed when the Context Group is made visible.

Table Layout

DiretX Only

Table Layout divide a control in to rows and columns similar to a grid layout or table in Microsoft Word.

Table Column

Used to define the width of columns as part of a table layout.

In the example below there are 3 columns. The first is 25 pixels wide. The two remaining columns are proportional and share the remaining space equally.

Define_Com Class(#prim_tblo) Name(#TableLayout) Define_Com Class(#Prim_tblo.Column) Name(#Column1) Parent(#TableLayout) Units(Pixels) Width(25) Define_Com Class(#Prim_tblo.Column) Name(#Column2) Parent(#TableLayout) Define_Com Class(#Prim_tblo.Column) Name(#Column3) Parent(#TableLayout)
Units

In conjunction with the Width/Height properties determines the size of the column/row.

Pixels

The column/row will be as wide/high in pixels as the value specified in the Width/Height property.

Proportion

The size is defined as a proportion of the available remaining space after columns/rows with a fixed size are evaluated.

In the example below, Column1 would use 40% of the available space while Column2 would use 60%

Define_Com Class(#Prim_tblo.Column) Name(#Column1) Parent(#TableLayout) Width(2) Define_Com Class(#Prim_tblo.Column) Name(#Column2) Parent(#TableLayout) Width(3)

Width

In conjunction with the Units property determines the width of the column. Width is numeric and represents either specific number of pixels or a proportion of the available space.

Table Row

Used to define the height of rows as part of a table layout.

In the example below there are 3 rows. The first is 25 pixels wide. The two remaining columns are proportional and share the remaining space equally.

Define_Com Class(#prim_tblo) Name(#TableLayout)

Define_Com Class(#Prim_tblo.Row) Name(#Row1) Parent(#TableLayout) Units(Pixels) Width(25)

Define_Com Class(#Prim_tblo.Row) Name(#Row2) Parent(#TableLayout)
Define_Com Class(#Prim_tblo.Row) Name(#Row3) Parent(#TableLayout)

Height Column

In conjunction with the Units property determines the height of the row. Height is numeric and represents either specific number of pixels or a proportion of the available space.

Table Item

Used to determine the size and position of the control being managed by the item.

Column

Reference to a Column (#prim_tblo.Column)

In conjunction with the ColumnSpan property, defines the start position and horizontal extent of the control managed by the item.

ColumnSpan

In conjunction with the Column property, defines the start position and horizontal extent of the control managed by the item.

Row

Reference to a Row (#prim_tblo.Row)

In conjunction with the RowSpan property, defines the start position and vertical extent of the control managed by the item.

RowSpan

In conjunction with the Row property, defines the start position and vertical extent of the control managed by the item.

Touch

Touch Properties Touch Events Touch Parameters

Touch Properties

Touch Scroll Property Touch Move Property Touch Rotate Property Touch Scale Property

Touch Scroll Property

DirectX Only

Determines how list controls such as Listview and Grid will respond when swiped.

Both

The list will scroll both vertically and horizontally.

Horizontal

The list will only scroll control horizontally.

Parent

Default – All touch move processing will be directed to the parent control. Touch events will be processed as normal mouse events causing selection and focus to fire.

Vertical

The list will only scroll control vertically.

Touch Move Property

DirectX Only

Determines how the control will respond to touch move processing.

Touch movements will directly affect the Top and Left properties of the control.

Both

The control can be moved both vertically and horizontally.

Horizontal

The control can only move along the horizontal axis.

Parent

Default – All touch move processing will be directed to the parent control.

Vertical

The control can only move along the vertically axis.

Touch Rotate Property

DirectX Only

Determines how the control will respond to touch rotation.

Touch rotation will directly affect the Rotation property of the control.

Yes

The control can be rotated using touch.

Parent

Default – All touch rotation processing will be directed to the parent control.

Touch Scale Property

DirectX Only

Determines how the control will respond to touch scaling.

Touch scaling will directly affect the ScaleHeight and ScaleWidth poperties of the control.

Yes

The control can be scaled using touch.

Parent

Default – All touch scaling processing will be directed to the parent control.

Touch Events

Touch Start Event Touch End Event Touch Change Event

Touch Start Event

DirectX Only

Fired when the user first makes physical contact with the screen.

Touch End Event

DirectX Only

Fired when the user stops touching the screen.

Touch Change Event

DirectX Only

Fired repeatedly between the TouchStart and TouchEnd boundaries to provide details of the touch events.

Having access to the details of a touch event allows the developer to control the nature of the touch processing. For example, the following code stops an image being moved beyond the bounds of its parent control.

```
Evtroutine Handling(#Image.TouchChange)
```

```
#Image.Top := #Image.Top.Bound( 0 (#Com_owner.Height - #Image.Height)
)
#Image.Left := #Image.Left.Bound( 0 (#com_owner.Width - #Image.Width) )
```

Endroutine

Touch Parameters

MoveLeft Parameter MoveTop Parameter ScaleHeight Parameter ScaleWidth Parameter Rotation Parameter Continue Parameter

MoveLeft Parameter

DirectX Only

Returns the number of pixels moved horizontally for each instance of the TouchChange event.

MoveTop Parameter

DirectX Only

Returns the number of pixels moved vertically for each instance of the TouchChange event.

ScaleHeight Parameter

DirectX Only

Returns the percentage scale height change for each instance of the TouchChange event.

ScaleWidth Parameter

DirectX Only

Returns the percentage scale width change for each instance of the TouchChange event.

Rotation Parameter

DirectX Only

Returns the rotation change in degrees for each instance of the TouchChange event.

Continue Parameter

DirectX Only

Allow touch processing to be stopped if required.

True

Default – Processing will continue as normal.

False

Touch processing will be halted.