Working with the Laszlo Component Set
An Overview for User Interface Designers

Components Overview

Laszlo Components (LZ Components) are a core set of customizable user interface devices included with Laszlo Presentation Server 2.0 (LPS 2.0). LZ Components are written in the Laszlo language (LZX) and designed to facilitate the construction of LZX applications. The set (illustrated on the following page) mimics the rich behavior of operating system and client software interface elements.

Component features

- Multiple states (up, mouse-over, down and disabled)
- Colorization
- Custom font
- Customizable spacing
- Layout options
- Keyboard navigation
- Automatic sizing based on content

Using Components

Components can be used in their default form, or may be modified to adapt to the design of the applications that include them. This document covers several levels of modification:

Colorization

The simplest method of modifying components is to change the “style”**.** Style is a palette which defines a family of colors for the component set. There are 6 styles included with LPS 2.0. New styles can easily be included by modifying existing values.

Modifying Attributes

Each component has a set of adjustable parameters called “attributes”. Attributes are contained in the component code and surface common controls for adjusting values such as width, height, color, and positioning. Information about the attributes of each component are located in the LZX documentation.

Replacing Visual Resources (Skinning)

Visual transformation of the components is possible by replacing resources (skinning). Knowledge of component construction, details of assembly and production methods are necessary to accomplish this.

Laszlo Components Set These are examples of the components included with Laszlo Presentation Server 2.0 (LPS 2.0).

Button

Button

Radio Button

radio button

radio button

Check Box

checkbox

checkbox

Scrollbar

Edit Text

Enter Text Here

List

Item One
Item Two
Item Three
Item Four

Combobox (Non-editable)

Choose One

Combobox (Editable)

Choose One

Combobox (Open)

Choose One
Sarah Allen
Peter Andrea
Bret Simister
Christian Walker

Menu

View File Document
Item One
Item Two
Item Three

Tabs

Tab One Tab Two

Tab Slider

Tab Element 1
Tab Element 2
Tab Element 3

Window with Scrollbar

Window

Window with Menu

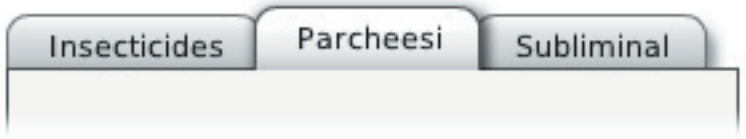
Window
View File Document
Item One
Item Two
Item Three
Item Four
Sub item One
Sub item Two
Sub item Three

Colorization

The color of the components can be globally modified by changing the style. Style is referenced by the application utilizing the component set and automatically affects any component included. There are six styles included in the Laszlo component set. The default style is “white”.

Included Styles

White (default)



Silver

Purple

Blue

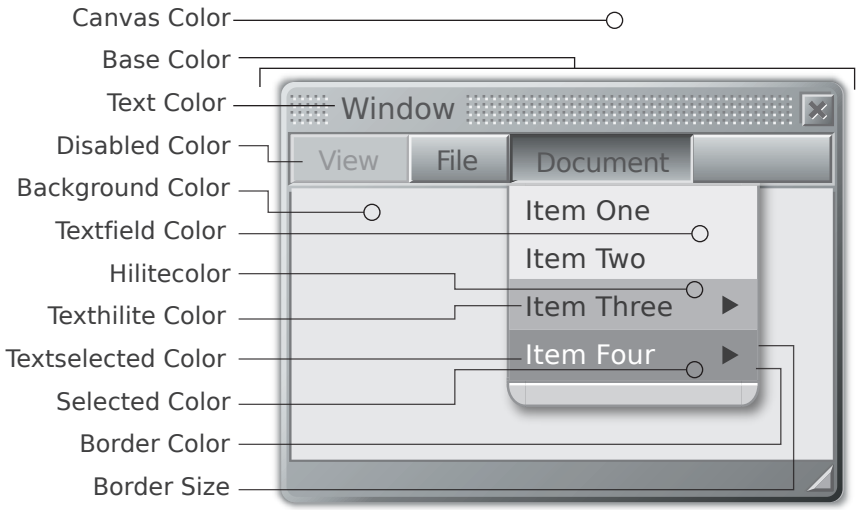
Green

Gold

Style

The set of attributes in a style specify the color values for different parts of a component. These colors can be adjusted independently and automatically affect the component set globally. Each of the style attributes below (except “Border Size”) is defined by a hexadecimal color value. Most of the colors used directly reflect the specified color. The “Base Color” attribute is the main tint applied across components. This value adds color to the resources which have been created in grayscale. This colorization will affect resources differently based on their original value. Some experimentation may be necessary to find a tinting color that produces the desired colorization. >view example<

Style Attributes



Style Attributes in Code

The styles are defined in the file “defaultstyles.lzx” which is located in the “base” directory (more on file locations in the “Component Files” on page 5). The code is fairly simple to evaluate and uses names for color values which are defined in the component color palette. An overview of the code for style is in the LZX reference.

Color Palette

It is helpful to create a color palette when building new styles. A palette is a simple library where hexadecimal values are given names. It is useful to experiment and choose multiple colors before creating a palette.

These are the colors defined in LZX and LZ Components. They can be referred to by name in LZX code. Component colors are in the “colors.lzx” located in the “base” directory.

LZX Default

- black #000000
- green #00800
- silver #C0C0C0
- lime #00FF00
- gray #808080
- olive #808000
- white #FFFFFF
- yellow #FFFF00
- maroon #800000
- navy #000080
- red #FF0000
- blue #0000FF
- purple #800080
- teal #008080
- fuchsia #FF00FF
- aqua #00FFFF

Component Default

- offwhite #F2F2F2
- gray10 #1A1A1A
- gray20 #333333
- gray30 #4D4D4D
- gray40 #666666
- gray50 #F7F7F7
- gray60 #999999
- gray70 #B3B3B3
- gray80 #CCCCCC
- gray90 #E5E5E5

- iceblue1 #325693
- iceblue2 #5381CE
- iceblue3 #BAC4D5
- iceblue4 #D5E4F3
- iceblue5 #EEF1F5

- palegreen1 #417641
- palegreen2 #B3D3B3
- palegreen3 #C0D49D
- palegreen4 #D3EAAA
- palegreen5 #EFF1E8

- gold1 #8E6409
- gold2 #CBB14B
- gold3 #E6CF74
- gold4 #F8E496

- sand1 #D4C6A1
- sand2 #D9D7C2
- sand3 #F2EEDB
- sand4 #F3EEE1

- ltpurple1 #645698
- ltpurple2 #B7B0D1
- ltpurple3 #CBC1ED
- ltpurple4 #E9E7F0

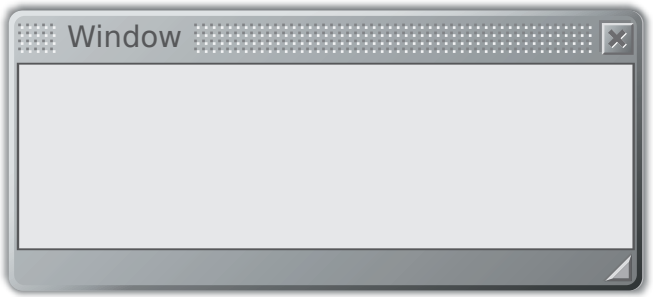
- grayblue #BEC2C8;
- graygreen #C0CCC0
- graypurple #9F9DB1

- ltblue #DDDDFF
- ltgreen =#DDFFDD

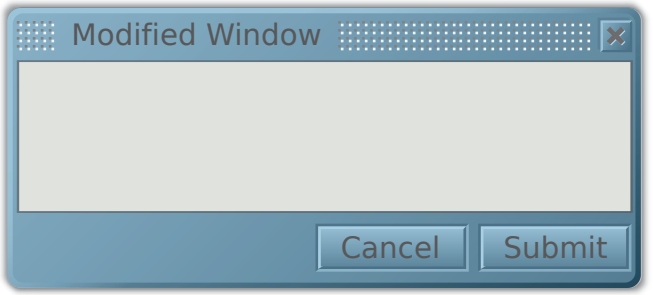
Modifying Attributes

Beyond the global adjustments achieved through styles, each component also has its own distinct attributes. While it is not necessary for a designer to program in LZX, understanding the basic syntax and an awareness of existing attributes expedites modification of the component set. The attributes of each component are listed in the LZX Reference.

A component is defined in code by a “view” or “tag”. Using the tag <window/>, for example, will add a window to an application. Window has a content and title area (with gripper) by default. The example below also has a title defined and enabled close and resize buttons.

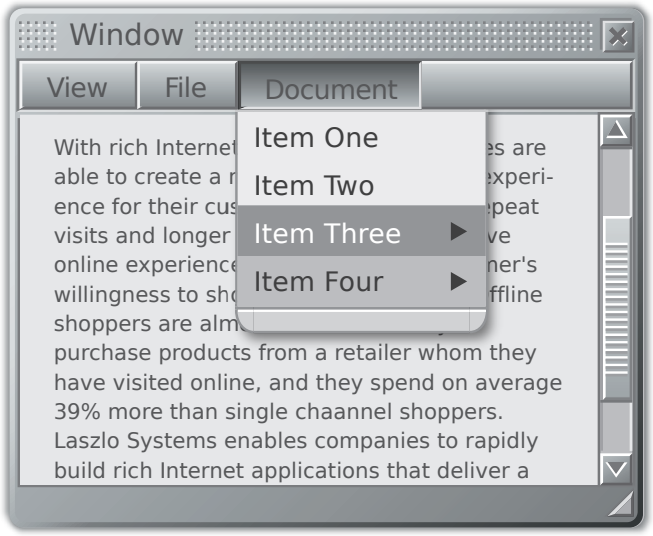


Attributes are used to describe and modify a particular instance of a component. When the attributes are accessed through the tag, that instance is modified. In the example below, the attribute for color is changed, the title has been revised, the offset for the content area has been increased, resize widget has been removed and buttons (with their own attributes) have been included.



This is only one of many possible ways to modify this particular component. Familiarity with attributes makes minor customization painless and expands the potential for creative solutions.

Components are designed to work together to form more complex structures. It is easy to add a scrolling list and a menu to a window simply by nesting the subordinate component tags in the window. The example below is constructed using only default components.



This flexibility and the ease of changing the values of attributes simplifies the modification of the default look for designers and developers.

Replacing Visual Resources (Skinning)

The look of a component can be significantly altered to correspond with the design of an application by changing resources. This is achieved most easily by replacing the default resources with custom resources of the same size, file name and format. Resources which are replaced, but contain the same name will automatically be updated when the application is refreshed. Alterations using sizes or file types different from the default set is possible, but will require a deeper understanding of construction methods. The Component Construction section of this document provides descriptions of how each is assembled, the file names, and states. If designs for an application involve a look + feel or interactivity beyond the scope of the default component set, the LZ and/or Base components will need to be modified.

Component Files

Component files are located in the LPS 2.0 directory. The path to the "Components" directory is: LPS2.0/WEB-INF/lps/components/... The Laszlo component set is comprised of four categories of files:

Base Components

LZ Components extend one or more base components. These provide the underlying logic for the set and contain no visual resources. For the most part, a designer will not need to manipulate these files.

LZ Components

LZ Components define the class for the UI device. These files utilize visual resources and define attributes of the component. These are the LZX files which will most likely be modified by a designer. They generally contain resource inclusions and attributes at the top of the code and are relatively easy to understand and manipulate.

LZ Resources

Resources are the visual assets that make up the look (skin) of a component. Resources may be a variety of file types, depending on the behaviors of the objects, the desired effect, or look. The resources are often pieces used to create whole components and their respective states. Visual reference for resources are covered in the "Component Construction" section beginning on page 7. The default color of the resources is gray, which is used to provide a neutral palette for colorizing. There are some visual elements in the components that are not external file resources. The LZX language is capable of building primitive shapes (rectangles) and these are used where possible within components. Often these shapes are more efficient to use than a resource as they decrease load time of an application and increase performance.

Fonts

A collection of various True Type Format (ttf) fonts are included with LPS 2.0. The default fonts used in components are the outline font Bitstream "Vera" and the Ultra Pixel Font "Verity". The outline font can be used at any point size, but is not recommended for sizes smaller than 12 point due to poor rendering quality of the Flash player. To circumvent this problem, the Verity is used for smaller instances. Ultra Pixel Fonts, unlike other bitmap fonts, forces anti-aliasing which creates smooth, legible characters. Verity is included in 9 and 11 point sizes with roman, italic, bold and bold italic styles. This font is designed to mimic the look of Bitstream Vera and can, therefore, be used seamlessly in conjunction with the outline version. Bitmap (or Pixel) fonts such as "Verity" will only render correctly when specified at 8pt. The Verity ttf file (9 or 11) must be used to change size. The default font Verity contains an unaccented

subset of characters. For the full Windows Western Unicode font set, use VerityPlus, which includes accented European characters.

The Bitstream font Vera was released as an open source font for the Gnome Project. The Ultra Pixel font Verity was built by Christopher Lowery using technology produced by Truth in Design.

Creating Resources (Tips + Tricks)
Creating your own resources to modify the look of a component requires an understanding of not only the construction, but also of the characteristics of file types and the manner in which LZX utilizes them.

Component File Types
Although LZX supports multiple graphic formats, The default resources for components are almost entirely .swf files. There a few instances where .pngs have been used for specific reasons (documented in Component Construction). The primary reason for using .swf files is that they are vector-based, and therefore resolution independent. This independence allows files to be resized or stretched without degrading. .Swf files are the only vector file type supported by LPS 2.0. Although the obvious tool for creating .swf files is Macromedia Flash, many vector-based applications, such as Adobe Illustrator allow export of files to this format. If the option is given, .swf files should always be saved as version 5.0. The exporter for Illustrator 10 does not provide an option, but uses 5.0 by default.

Working with Vector Art
There are some obstacles to overcome when creating vector art for export. A sometimes useful but quirky effect is that, unlike bitmaps, vector shapes can use fractions of pixels as values for placement and dimension. This can cause problems when the shape is exported and used in LZX. The shape will anti-alias to compensate for the non-integer value and effectively change the dimension of the resource. There are a handful of instances where this can be beneficial, but most often it will cause annoying alignment issues.

Working on a Grid
There are several methods to ensure integer pixel values. Regardless of the vector-based application being used, it is crucial to set the unit of measure to pixels (or points, as they are equivalent). This includes creating a pixel grid and using the “snap to grid” feature. It is also helpful to control dimension and position objects numerically, as this will lower the possibility of fractional values. These procedures predominantly apply to rectilinear objects. Circles and angles will anti-alias by nature, however, by aligning points on pixel boundaries more predictable anti-aliasing will be experienced. There are also instances (such as in Macromedia Flash) where the position of an object on the page or stage will be translated into the resulting file. Unless an offset is part of the desired effect, make sure that objects are “zeroed out” (x=0, y=0).

Transparency
Many of the resource files employ transparency. This allows the object to be placed on, and anti-alias to any background. This is an automatic feature of .swf files. Unless a background is created or specified, areas around the object are automatically transparent. In “multi-frame” resources (those with multiple states), transparency is used to ensure alignment of resources with different dimensions. The disabled state of most components is an example of this. Resources for this state are smaller than the enabled states, but are aligned because they incorporate a transparent rectangle with dimensions equal to the enabled resource.

Preparing Resources
To create the multiple resources that are used to build the majority of the components it is useful to first build the object as a whole and then slice it. Some vector-based applications (including Adobe Illustrator) have the ability to “slice to guides” this is intended to be used in the production of multi-part resources, but does not work reliably. A superior, but less-automated method is to cut along the grid or guides with the “knife” tool. This is handy because it allows cutting through multiple (unlocked) layers.
To guarantee a straight cut along pixel boundaries set your document’s magnification to increments of 100 (200%, 1600% etc) and hold shift key to force a straight line. With the resource cut into appropriate pieces, it is then necessary to move each resource individually to a separate document and export to .swf.

Objects with solid fills can be easily cut and exported from any vector based application. Objects with strokes will not be correctly rendered in LZX and should be converted to outline shapes (Strokes created in Macromedia Flash are an exception). If the object to be cut has a gradient fill, many applications will not cut the fill correctly. For objects with gradient fills it is preferable to export the object whole, import it into Flash for cutting. Flash effectively “flattens” the resource and will not repeat the gradation in each piece. When importing vector files into Flash, it is best to already have them in the .swf format. Cutting and pasting or importing Illustrator files into Flash has a tendency to drop pieces, botch bezier curves, incorrectly translate color or entirely deny the import.

Component Construction
In order to successfully skin the component set without major modification of the code it is imperative to understand the methods of construction. The following pages detail each component.

Button (filename: button.lzx)

>View Example< >View Documentation<

The default component button has a height of 22 pixels and a width determined by the length of the button text and its padding. The button face is 18 pixels tall with 2 pixel borders for the bezels. The type is centered on the button face. The button is comprised of scalable vector resources which enable disproportional resizing without distortion. Resizing the button will not alter the text size. The button can also contain an icon image with or without text included (see Button example in the LPS 2.0 Welcome Page).

The button form is constructed of four views. The views are:
1. The “Outer Bezel” creates an indented area around the button
2. The “Inner Bezel” defines the edges of the button
3. The "Simple Face" is the face of the button.
4. The "Text" for the button label -11 pt Verity Roman
   (An icon or image can be used with text or alone.)

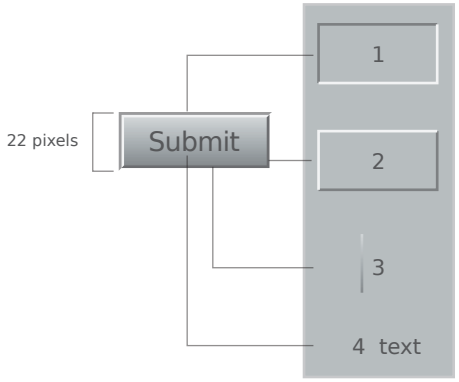
Views 1-3 are constructed as multiple frame resources which change based on the state of the button.

The outer and inner bezels (and each of their states) are constructed of a “hairline” .swf file with two different colored strokes on each rectangle. These files were built using the Flash authoring tool, and take advantage of a feature of the .swf file format that is not supported in other vector-based programs (such as Adobe Illustrator). Hairline strokes created in Flash will retain their weight regardless of scaling*. This means that using a size other than the default will not distort the look of the button. Although transparent, the disabled state of the button (which is visually smaller) retains the Outer Bezel to insure alignment with the other resources.

The face view contains a .swf resource which is a vector gradient acting as the face of the button. There are 4 separate files for each of the button states, each with its own gradation and opacity. These files are saved as 1 pixel wide by 18 pixel high (default face height) resources and are scaled horizontally to the width of the button.

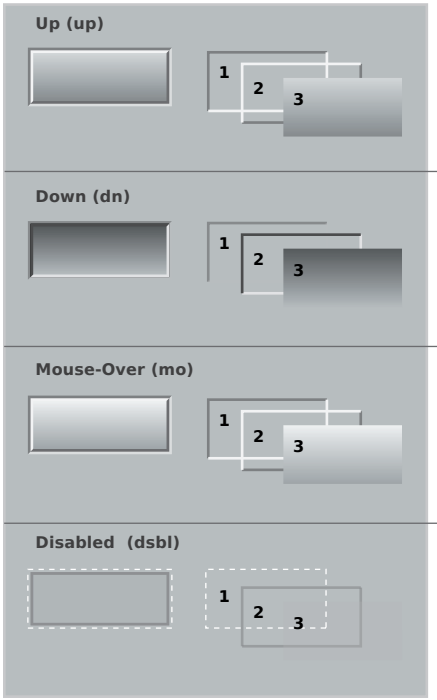
*An unfortunate effect of this technique is that a “hairline” only retains its width when decreasing scale. Scaling a “hairline” up will scale the thickness of the stroke. In an effort to work around this, the assets for both outer and inner bezels (and their states) are set at the overly large size of 500x500 pixels.

Construction



1. Outer Bezel: “hairline” .swf file
2. Inner Bezel: “hairline” .swf file
3. Simple Face: gradient swf. (stretches to fit width)
4. Text: 11pt Verity Roman

States and Resources



Button Up
1. Translucent Outer Bezel (file: bezel_outer_up.swf)
2. Solid Inner Bezel (file: bezel_inner_up.swf)
3. Disabled Face (file: simpleface_up.swf)

Button Down
1. Translucent Outer Bezel (file: bezel_outer_dn.swf)
2. Solid Inner Bezel (file: bezel_inner_dn.swf)
3. Disabled Face (file: simpleface_dn.swf)

Button Mouse-Over
1. Translucent Outer Bezel (file:bezel_outer_mo.swf)
2. Solid Inner Bezel (file: bezel_inner_mo.swf)
3. Disabled Face (file: simpleface_mo.swf)

Button Disabled
1. Transparent Outer Bezel (file: transparent.swf)
2. Solid Inner Bezel (file: outline_dsbl.swf)
3. Disabled Face (file: simpleface_dsbl.swf)

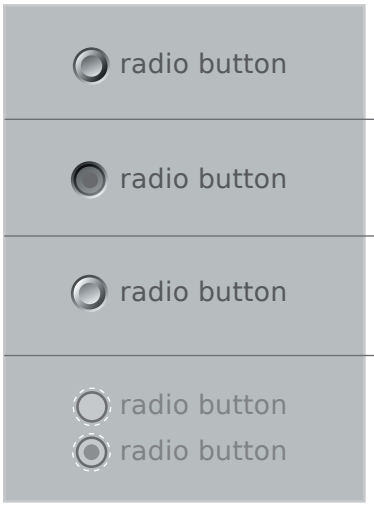
Radio Button (filename: radio.lzx)

>View Example< >View Documentation<

The default radio button has a height and width of 14 pixels. The button is a scalable vector resource which can be proportionally resized without distortion. Disproportional scaling will result in an oval button.

Radio buttons are constructed as single resources for each of the various states. The placement of text associated with a radio button has been defined to compliment the default size. Text is positioned to the right of the button. The disabled state, although visually smaller, has the same dimensions as the enabled versions. A transparent circle (indicated by white dashed line) ensures alignment when a button changes state.

States and Resources



Radio Button Up (up)
1. Radio up (file: radiobtn_up.swf)
2. Text: 11pt Verity Roman

Radio Button Down (dn)
1. Radio down (file: radiobtn_dn.swf)

Radio Button Mouse-Over (mo)
1. Radio mouse-over (file: radiobtn_mo.swf)

Radio Button Disabled (dsbl)
1. Radio unselected + disabled (file: radiobtn_dsbl_up.swf)
2. Radio selected + disabled (file: radiobtn_dsbl_dn.swf)
3. Text: disabled color

dashed line indicates actual boundaries of file

Check Box (filename: checkbox.lzx)

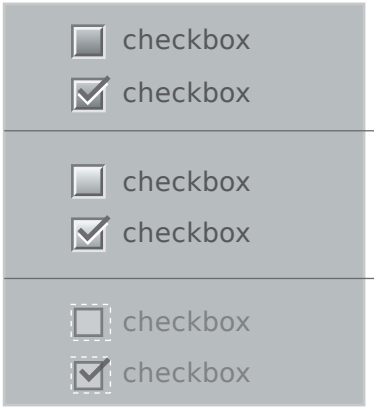
>View Example< >View Documentation<

The default check box has a height and width of 15 pixels. The check box is a scalable vector resource which enables proportional resizing without distortion. Disproportional scaling will result in an odd check mark.

All resources in check box (including disabled state) have the same exterior dimensions. Resources which appear to be smaller contain invisible rectangles that surround the visible art. This ensures the resource frames automatically register with one another, and will not need to be repositioned in code.

Check box is constructed as single resources for the various states. The placement of associated text has been defined to compliment the default size. Text is left-aligned and set to the right of the button. The disabled state, although visually smaller, has the same dimensions as the enabled versions. A transparent rectangle (indicated by white dashed line) insures alignment when

States and Resources



Check Box Up (up)
1. Check box unselected (file: checkbox_off.swf)
2. Check box selected (file: checkbox_on.swf)

Check Box Mouse-Over (mo)
1. Check box unselected (file: checkbox_off_mo.swf)
2. Check box selected (file: checkbox_on_mo.swf)

Check Box Disabled (disable)
1. Check box unselected (file: checkbox_disable_off.swf)
2. Check box selected (file: checkbox_disable_on.swf)
3. Text: disabled color

dashed line indicates actual boundaries of file

Floating List (filename: floating list.lzx)

>View Example< >View Documentation<

The floating list is used in conjunction with other components. Both combobox and menu use this component.

The floating list is defined by six pieces:
1. The end caps (constructed of 3 pieces each)
2. The menu area
3. List highlight
4. List selection
5. Text (icon can be used in place of text)
6. The menu shadow

The menu, highlight and selection areas are shapes defined in LZX code and have no external resources. The border and background colors are attributes which are defined in the floating list class and styles; both can be easily customized. The highlight and selection are created by telling the list item background color to change depending on interaction.

The end caps are rarely used together on one floating list and are each comprised of three .swfs: left, middle and right. The end caps are designed to scale horizontaly. Scaling the end caps vertically is not advised. Text is 11pt Verity Roman

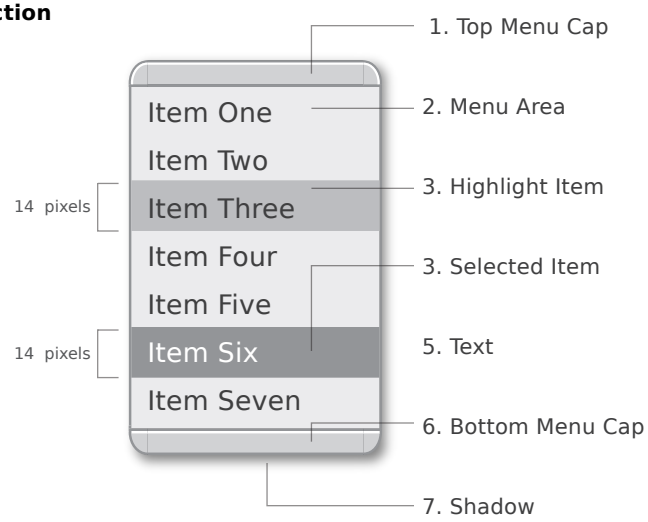
The highlight and selected rectangles have a default height of 19 pixels and a width determined by the width of the menu. The type is inset and left aligned within the menu.

The menu shadow is a 5 piece view*. The shadow middle resources stretch in both the direction of their orientation, and are constrained to the width and height of the floating list. Shadows were constructed in Macromedia Flash using "Soften edges" and a gradation of 100% black to 0% black.

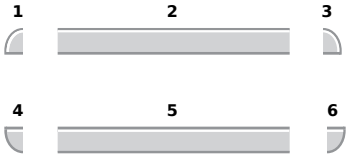
The top right shadow resource has 3 frames that switch between states depending on the context and positon of the menu
The frames are:
“pop_shadow_flush_top_rt.swf" no top cap (combo box, menu)
”pop_shadow_corner_top_rt.swf no top cap (submenus)
“pop_shadow_oval_top_rt.swf" for use with the top cap

*More info on multi-piece views:

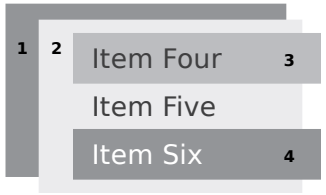
Construction



Resources



End Caps
1. Left end - fixed width (file: topmenu_lft.swf)
2. Mid - stretches width* (file: topmenu_mid.swf)
3. Right end - fixed width (file: topmenu_rt.swf)
4. Left end - fixed width (file: botmenu_lft.swf)
5. Mid - stretches width* (file: botmenu_mid.swf)
6. Right end - fixed width (file: botmenu_rt.swf)



Menu Area, Highlights and Selection
LZX rectangles
1. Border view
2. Background view
3. Highlight color of listitem
4. Selected color of listitem

Shadow
1. file: pop_shadow_oval_top_rt.swf
2. file: pop_shadow_mid_rt.swf
3. file: pop_shadow_bot_lft.swf
4. file: pop_shadow_bot_mid.swf
5. file: pop_shadow_bot_rt.swf

* actual file width is 1 pixel

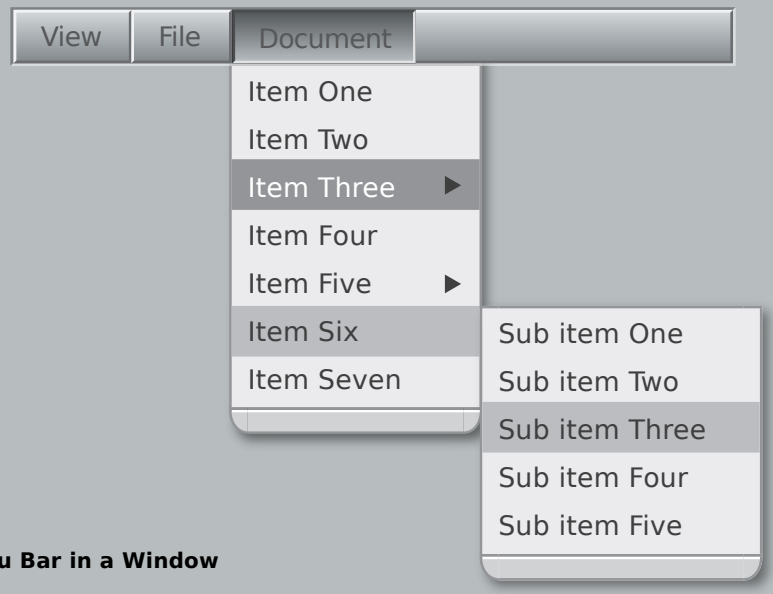
Menu (filename: menu.lzx)

>View Example< >View Documentation<

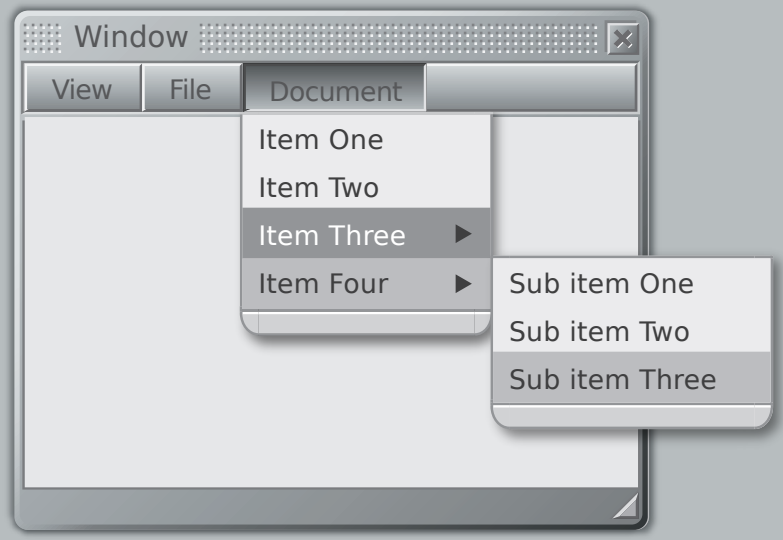
Both menu and menubar combine the components "Button" and "Floating List" to form a menuing system. The menu buttons and menu bar use the "button" component. The menu list itself is a "floating list", which is itself an extension of "list". Modifying these components will cause the same changes to be inherited by the menu. These components have been integrated and coded to have the behavior of a menubar or menu.

Menu is an constructed of four components :
1. Menu button - an extension of button
2. Menu bar - an inactive bar utilizing the resources for button
3. Menu list - an extension of floating list
4. Text - contained in an input text field

Menu Bar and nested menus



Menu Bar in a Window



Scroll Bar (file name: scrollbar.lzx)

>View Example< >View Documentation<

The scroll bar has separate classes and resources for each axis. For the purpose of simplicity, only the vertical version (aka Scroll Y) is covered in this document.

The scroll bar is defined by six pieces:
1. Scroll background
2. Up button
3. Scroll track
4. Scroll thumb (3 pieces)
5. Gripper
6. Down button

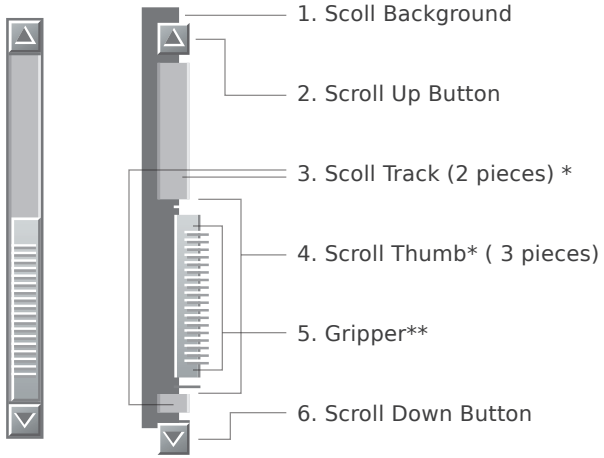
The scroll background is a colored LZX view that contains all of the resources. The view is offset to create a one pixel border around the scrollbar.

The scroll track is actually two views sandwiched between the up scroll button, the thumb and the down scroll button. The scroll track resizes as the thumb is moved to create the illusion of the thumb moving over it. The scroll track is a clickable resource which moves the thumb and scrolls the page. The scroll track has a down and disabled state, but no mouse-over.

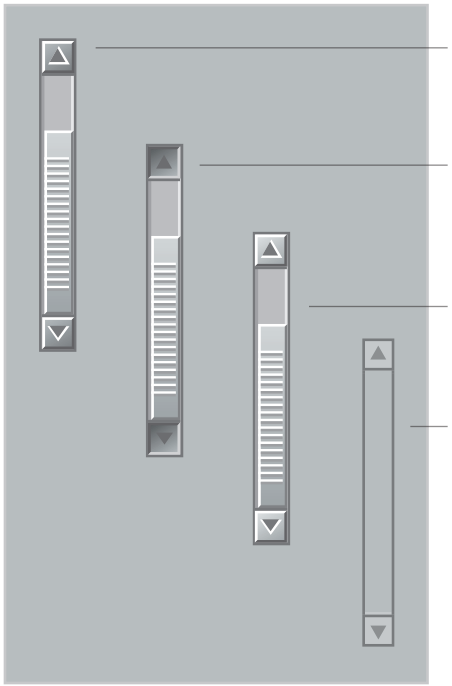
The scroll thumb is a set of three scalable vector resources. The scrollbar changes its height based on the amount of content clipped within the scrolling content area.

The gripper is a very long (440 pixels) .PNG resource that overlays the thumb and is centered with a fixed amount padding. It is clipped and constrained to the height of the thumb (minus the padding). As a .SWF resoure this file contains many points, and since it is not designed to scale, it is more efficient saved as a bitmap .PNG (approx. 60x smaller).

Construction



Resources (files that change)



Up Resources
1. Up button (file: scrollbtn_y_top_up.swf)
2. Down Button (file: scrollbtn_y_bot_up.swf)

Down Resources
1. Up button (file: scrollbtn_y_top_dn.swf)
2. Down button (file: scrollbtn_y_bot_dn.swf)
3. Scrolltrack (file: y_scrolltrack_dn.swf")

Mouse-Over Resources
1. Up button (file: scrollbtn_y_top_mo.swf)
2. Down button (file: scrollbtn_y_bot_mo.swf)

Disabled Resources
1. Up Button (file: scrollbtn_y_top_dsbl.swf)
2. Down Button (file: scrollbtn_y_bot_dsbl.swf)
3. Scrolltrack (file: y_scrolltrack_dsbl.swf")

*resource is actually 1pixel tall
**resource is actually 440 pixels tall

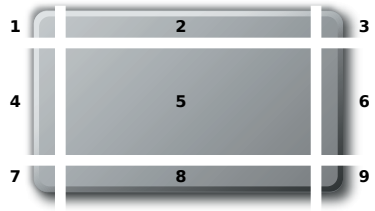
Window Panel (filename: windowpanel.lzx)

>View Example< >View Documentation<

Window panel is a fixed-size window. It is also the basis for “window” (which is resizable), “modal dialog”, and “alert”. The base art for “window panel” is constructed of 9 pieces. 4 corners, 4 resizeable middle edges and the middle. There are also controls included: the gripper (and title) and close button (with 4 states).

The window panel has 3 states: deslected (_dslct), selected (_slct) and dragging (_drag). The background of the deselected state has less contrast and no shadow. The selected state has more contrast and a small perimeter shadow. Dragging the window creates a more pronounced shadow, adding to the illusion that the window has been brought to front and is above other UI objects. Drag state only applies to resources on the right and bottom sides, the top and left use the same resources as the selected state.

Base Resources



Top

1. Left (file: top_lft_slct.swf)
2. Middle (file: top_mid_slct.swf)
3. Right (file: top_rt_slct.swf)

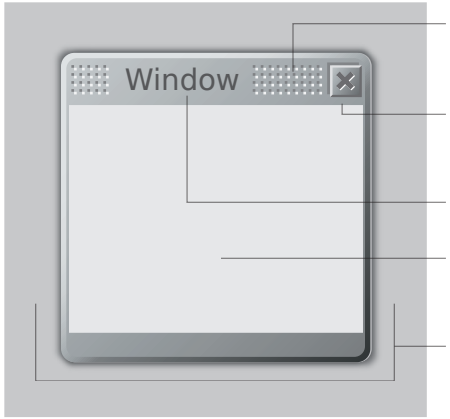
Middle

4. Left (file: mid_lft_slct.swf)
5. Middle (file: mid_mid_slct.swf)
6. Right (file: mid_rt_slct.swf)

Bottom

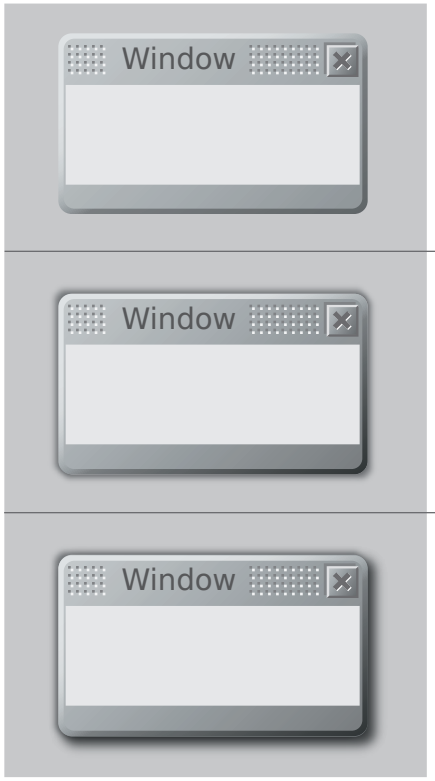
7. Left (file: bot_lft_slct.swf)
8. Middle (file: bot_mid_slct.swf)
9. Right (file: bot_rt_slct.swf)

Construction



1. Gripper: .png file with transparency (569 pixels long)
2 Close Button: four .swf files for the states
3. Title: default text: 11pt Verity Roman
4. Content: Colorized LZX view
5. Base: 9 piece resizeable view window with gradient and shadow

States



Window Panel Deselect

Window Panel Select

Window Panel Dragging

Tab Slider (filename: tabslider.lzx)

>View Example< >View Documentation<

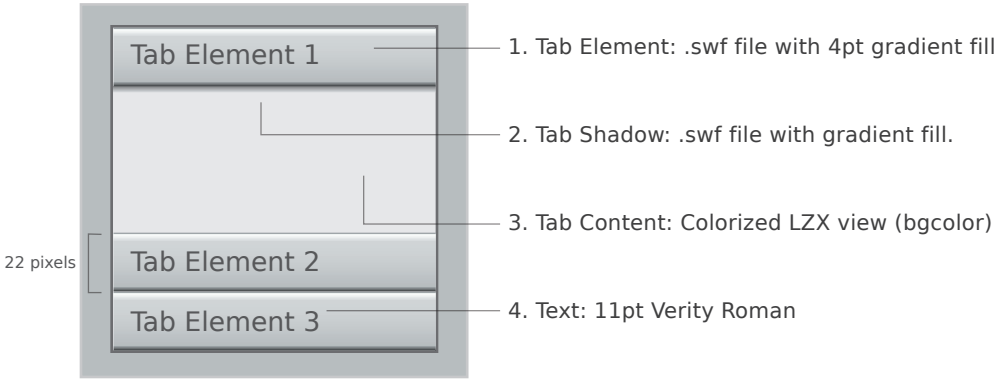
The default tab element (horizontal bar in tab slider) has a height of 22 pixels and a width determined by the width of the tab slider. The type is left aligned on the tab element with an inset. The tab element is comprised of a scalable vector resource which enables disproportional resizing without distortion. Resizing the button will not alter the text. Changing the height of the tab element will extend the gradient (more highlight, more shadow) causing it to look more cylindrical as it grows.

The button form is constructed of four views:
1. The “Tab Element” a single .swf resource with a gradient
2. The “Tab Shadow” resides below the open tab
3. The "Content" area
4. The "Text" view (can also be used with an icon)

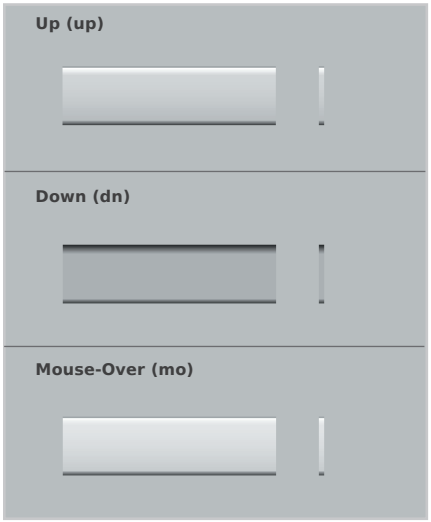
The tab element uses three resources for the states (up, mouse-over and down)

The content view is the interior of the tab element. This area is inset so that any objects placed in the content area has a margin.

Construction



States and Resources



Tabslider Up
1. Tab element (file: tab_slider_up.swf)

Tabslider Down
2. Tab element (file: tab_slider_dn.swf)

Tabslider Mouse-Over
3. Tab element (file: tab_slider_mo.swf)

Tab Panel (filename: tabs.lzx)

>View Example< >View Documentation<

The default tabs have a height of 26 pixels and the width is determined by the length of the tab text and its padding. Although the resources appear to be different sizes, the selected and deselected resources are actually the same height. The deselected pieces have a transparent layer that makes up the difference. The type is centered on the tab face and changes its position programatically. The tab is comprised of scalable vector resources which enable proportional resizing without distortion. resizing the button will not alter the text size or position.

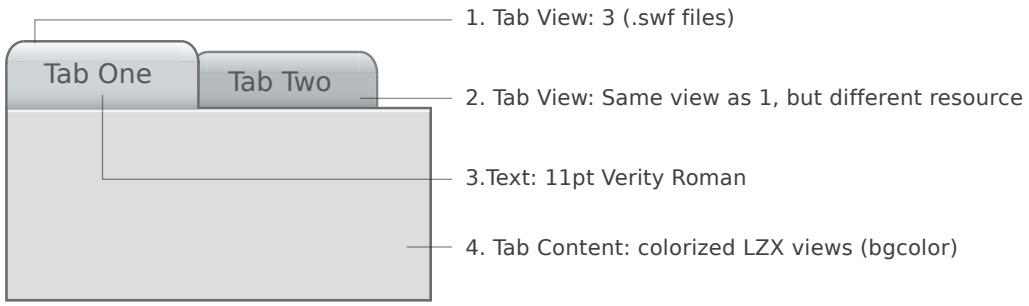
The tab is constructed of five views:
1. The left side of the tab
2. The middle stretchable section of the tab
3. The right side of the tab
4. The “border” of the content area.
5. The content area background

Each of the first three views contains multiple resource frames which change based on the state of the tab.

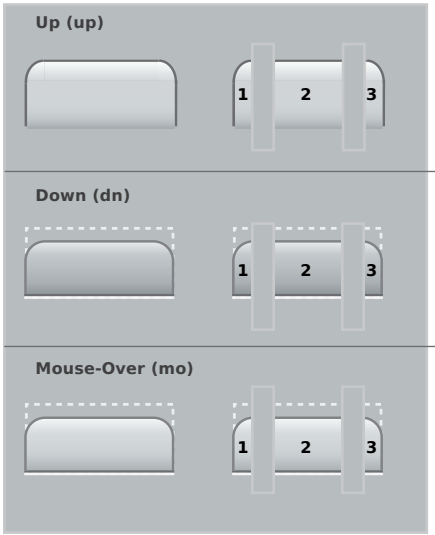
The left and right tab resources (and each of their states) are constructed of vector .swf files and are designed to be used at their current size. The middle piece is a vector .swf file constructed to scale horizontally as the tab width increases.

Th border and content area are both colorized LZX views. The default border is set at 1 pixel and retains that dimension as the content area grows. The content area has an 8 pixel inset on all sides, so that content is automatically inset.

Construction



States and Resources



Tab Panel Selected
1. Left Tab Edge(file: tab_slct_lft.swf)
2. Tab Middle (file: tab_slct_mid.swf)
3. Right Tab Edge (file: tab_slct_rt.swf)

Tab Panel Deselected
11. Left Tab Edge(file: tab_dslct_lft.swf)
2. Tab Middle (file: tab_dslct_mid.swf)
3. Right Tab Edge (file: tab_dslct_rt.swf)

Tab Panel Mouse-Over
1. Left Tab Edge(file: tab_mo_lft.swf)
2. Tab Middle (file: tab_mo_mid.swf)
3. Right Tab Edge (file: tab_mo_rt.swf)

Combo Box (filename: combobox.lzx)

>View Example< >View Documentation<

The combo box is defined by three pieces:
1. The button, which activates the menu (right side)
2. The text area, containing the default menu item
3. The menu (aka floating list)

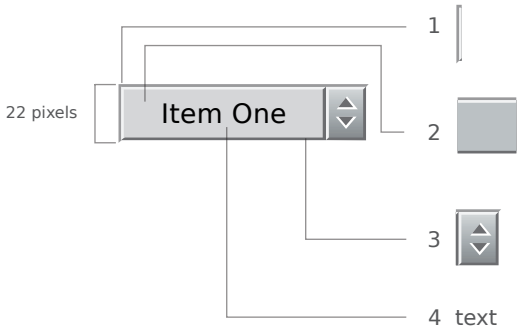
The combo box contains an attribute to create either editable or non-editable instances of this component. The button is the same for both versions and consists of a multi-frame resource for the different states. The text area has a different look for editable and non-editable versions. The floating list is the same for both.

The default combobox has a height of 22 pixels and a width determined by the length of the largest content item and its padding. The type is left aligned.

Non-Editable Combo Box

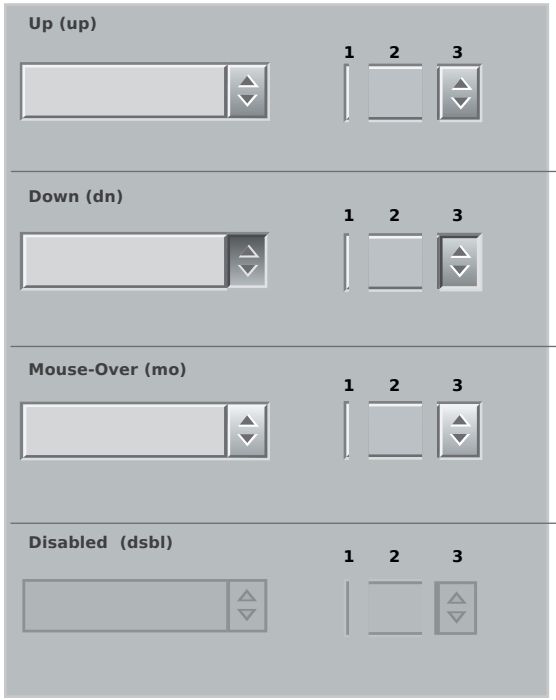
Editable Combo Box

Construction



1. Right End Cap: .swf with indent, hilite and shadow
2. Middle Section: stretchable .swf
3. Button: .swf file
4. Font: 11pt Verity Roman

States and Resources



Combobox Up
1. Left end - fixed width (file: popup_lft_up.swf)
2. Mid - stretches width* (file: popup_mid_up.swf)
3. Button (file: popbtn_rt_up.swf)

Combobox Down
1. Left end - fixed width (file: popup_lft_up.swf)
2. Mid - stretches width* (file: popup_mid_up.swf)
3. Button (file: popbtn_rt_dn.swf)

Combobox Mouse-Over
1. Left end - fixed width (file: popup_lft_up.swf)
2. Mid - stretches width* (file: popup_mid_up.swf)
3. Button (file: popbtn_rt_mo.swf)

Combobox Disabled
1. Left end - fixed width (file: popup_lft_dsbl.swf)
2. Mid - stretches width* (file: popup_mid_dsbl)
3. Button (file: popbtn_rt_dsbl.swf)

*actual file dimensions: 1 pixel wide, 18 pixels high

Editable Text Field (filename: edittext.lzx)

>View Example< >View Documentation<

The default edit text field has a height of 22 pixels and a width of 106 pixels. The field is comprised of scalable vector resources which enable disproportional resizing without distortion. Resizing the edit text field will not alter the text.

Edit Text is constructed of four views:
1. The “Outer Bezel” creates an indented area around the field
2. The “Inner Bezel” defines the recesss of the field
3. The "Face" is the background and center of the field
4. The "Text" view which is contained in an input text field

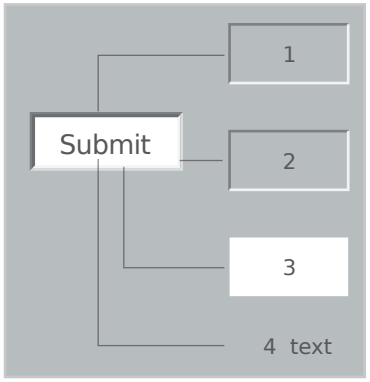
Each of the first three views contains multiple resource frames which change based on the state of the button.

The outer and inner bezels (and each of their states) are constructed of a “hairline” .swf file with two different colored strokes on each rectangle. These files were built using the Flash authoring tool, and take advantage of a feature of the .swf file format that is not supported in other vector-based programs (such as Illustrator). “Hairlines” created in Flash will retain their weight regardless of scaling*. This means that using a size other than the default will not distort the look.

The face is an LZX view with a background color (bgcolor). the opacity and bgcolor change between enabled and disabled states.

*An unfortunate effect of this technique is that a hairline stroke only retains its width when decreasing scale. Scaling a hairline up will scale the thickness of the stroke. In an effort to work around this, the assets for both outer and inner bezels (and their states)

Construction



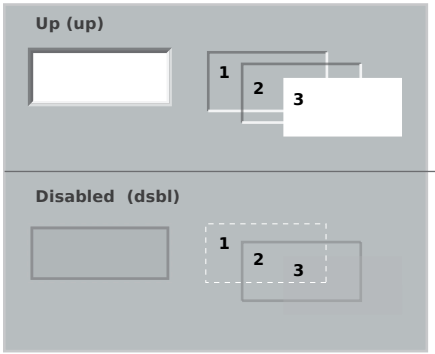
1. Outer Bezel: “hairline” .swf file

2. Inner Bezel: “hairline” .swf file

3.Text Field Face: LZX view (style name: textfieldcolor)

4. Text: 11pt Verity Roman

States and Resources



Text Field Enabled
1. Translucent Outer Bezel (file: bezel_outer_dn.swf)
2. Solid Inner Bezel (file: bezel_inner_dn.swf)
3. Face (view name: "_face")

Text Field Disabled
1. Translucent Outer Bezel (file: transparent.swf)
2. Solid Inner Bezel (file: outline_dsbl.swf)
3. Disabled Face (view name: "_face")

Focus (filename: basefocus.lzx in “base” directory)

>Documentation<

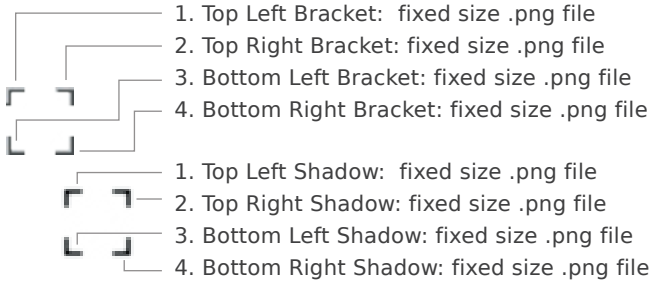
Focus is the visual component of keyboard navigation. Keyboard navigation allows the components to be used without a mouse. The tab key activates focus and moves between components in an application, the arrow keys enable selection within a component and the enter key or spacebar activates the selection. Focus appears as four brackets which adjust their dimension to surround the selected component. The brackets animate between objects as the tab key is pressed and disappears after a few seconds. Focus is enabled by default and has automatic awareness of included components and their position within an application.

Focus is defined by eight pieces:
1-4. The corner brackets
5-8. Shadows of the corner brackets

The bracket edges are two pixels thick (light and dark rows), enabling contrast against most background colors. Small drop shadows give the illusion that the brackets are floating slightly above the selected component. Focus has only one state, and can be disabled.

Construction

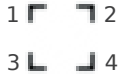
Focus Composite



Resources

Focus Brackets

1. file name: focus_top_lft.png
2. file name: focus_top_rt.png
3. file name: focus_bot_lft.png
4. file name: focus_bot_rt.png



Focus Shadows

1. file name: focus_top_lft_shdw.png
2. file name: focus_top_rt_shdw.png
3. file name: focus_bot_lft_shdw.png
4. file name: focus_bot_rt_shdw.png