Chapter 10 — Widget Encyclopedia

This section of the manual provides a reference description of the widgets that can be found on WindowBuilder Pro's tool palette, in alphabetical order. We have also provided a reference description of the **SubPane** class for convenience. Each description includes the following:

- the source of the widget: Objectshare or Digitalk
- an overview of the widget's functionality and general usage
- a list of the public protocol for the widget
- a list of the supported events for the widget
- a description of any WindowBuilder Pro extensions associated with the widget

The widgets that WindowBuilder Pro supports include the following:

- **ActionButton**

A button to which a specific action may be attached

- **AnimationPane**

Used to implement simple animation

- **Button**

A simple push button

- **CheckBox**

A toggle button used to represent a true or false state

- **CheckBoxGroup**

A CompositePane used to quickly implement groups of CheckBoxes

- **ComboBox**

A hybrid between an EntryField and a ListBox allowing for text entry or list selection

- **CPStaticGraphic**

Used to display a static graphic

- **DrawnButton**

A simple push button that may contain a graphic rather than a string

- **EnhancedEntryField**

An advanced single line entry control that provides character and field level validation

- **EntryField**

A simple single line text entry control

- **EntryFieldGroup**

A CompositePane used to quickly implement groups of EntryFields

- **GraphPane**
Provides for general drawing within a pane

- **GroupBox**
Used to visually group related controls

- **LinkButton**
A button used to automate linking windows together

- **ListBox**
Provides a general selection capability from a collection of mutually exclusive choices

- **ListPane**
A ListBox implemented entirely within Smalltalk

- **MultipleSelectListBox**
A ListBox allowing selection of multiple choices within a list

- **RadioButton**
A toggle button used to provide selection from mutually exclusive choices

- **RadioButtonGroup**
A CompositePane used to quickly implement groups of RadioButtons

- **ScrollBar**
A generic slider control

- **SexPane**
A CompositePane used to specify the sex of an individual

- **StaticBox**
Used to draw empty or filled rectangles

- **StaticText**
A static pane used for displaying text

- **TextEdit**
A multi-line text editor

- **TextPane**
A multi-line text editor implemented entirely within Smalltalk

- **ThreeStateButton**
A CheckBox with a third state to reflect ambiguity

- **WBToolBar**
Used to implement standard horizontal toolbars

SubPane Digitalk

SubPane is the ultimate abstract superclass of all widgets. It provides general behavior that applies to all of its subclasses. Before discussing each of the widgets in detail it is important to touch on this first.

Protocol

backColor
Return the background color of the widget.

backColor: *aColor*
Set the background color of the widget to *aColor*.

bringToTop
Bring the widget to the top of its overlapping siblings and activate it.

disable
Disable the widget.

disabled
Is the widget disabled?

enable
Enable the widget.

font
Return the font of the widget.

font: *aFont*
Set the font of the widget to *aFont.*

foreColor
Return the foreground color of the widget.

foreColor: *aColor*
Set the foreground color of the widget to *aColor*.

framingBlock: *aBlock*
Set the framing block of the widget to *aBlock* which, when executed, yields the pane frame rectangle. *aBlock* may either be a one argument block or, in the case of WindowBuilder, an instance of a FramingParameters object. For details on FramingParameters objects see **Appendix C**.

framingRatio: *aRectangle*
Set the framingBlock of the widget to a block which, when executed, yields the pane frame rectangle proportional with the ratios specified by *aRectangle.*

haveFocus
Does the widget currently have the focus?

hideWindow
Make the widget invisible.

invalidateRect: *aRectangle*
Force the region of the widget specified by *aRectangle* to be repainted. If aRectangle is nil, the entire widget will be redrawn.

owner: *anObject*
Set the owner of the widget to *anObject.* The owner of a widget becomes the target of any event handlers (through the **when:perform:** mechanism) specified for the widget.

paneName: *aString*
Set the name of the widget to *aString.* When in the context of a ViewManager, the widget may be retrieved with the following code: `self paneNamed: aString`.

rectangle
Return the bounding box of the widget.

setFocus
Set the focus to the widget.

setMenu: *aMenu*
Set the menu of the widget to *aMenu.*

setPopupMenu: *aMenu*
Set the popup menu of the widget to *aMenu.*

showWindow
Make the widget visible.

when: *anEvent* **perform:** *aSelector*
Notify the owner of the widget whenever *anEvent* occurs by performing *aSelector*. *aSelector* takes one argument, the widget itself, and must be a method which the widget's owner can understand.

Supported Events

display
This event occurs whenever the widget wants to display itself. This event is only received by GraphPane and its subclasses.

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

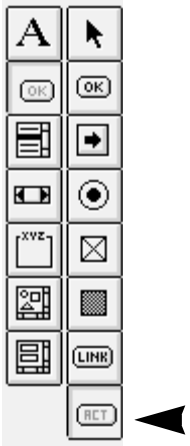
getMenu
This event occurs whenever a widget looks for its regular menu (as opposed to its popup menu). This event is not used in conjunction with WindowBuilder - use **getPopupMenu** instead.

getPopupMenu
This event occurs whenever a widget looks for its popup menu (generally as a result of a right button click).

help
This event occurs whenever the F1 key is pressed when the widget has focus.

resize
This event occurs after any widget is resized.

ActionButton Objectshare



ActionButtons provide a mechanism to attach, create, and reuse actions without having to write ViewManager methods. For example, a predefined "Cancel" action would know how to close the window to which it is attached. ActionButtons in conjunction with ActionMenus, LinkButtons, and LinkMenus provide the cornerstone of WindowBuilder Pro's rapid prototyping capabilities.

Since ActionButtons inherit from DrawnButton, they may take on the visual appearance of either a simple Button or may have a graphic attached to them. We have also provided an invisible style which can be used to create "sense regions" on other graphic images. For example, you could place invisible buttons on top of a large imported bitmap.

Protocol

action: *aSymbol*
Specify the action to be performed by the button when it is clicked. To get a list of available actions do the following: WBAction listActions inspect. New actions may be created with the ActionButton attribute editor.

cpContents
Return the label of the button or the file name in which its bitmap is stored.

cpContents: *aString*
Set the contents of the button. If *aString* is a valid bitmap file name (.BMP), the button will display the bitmap. If not, the button will look like a simple Button with *aString* as its label.

contents: *aBitmap*
Set the contents of the button to the bitmap specified by *aBitmap.*

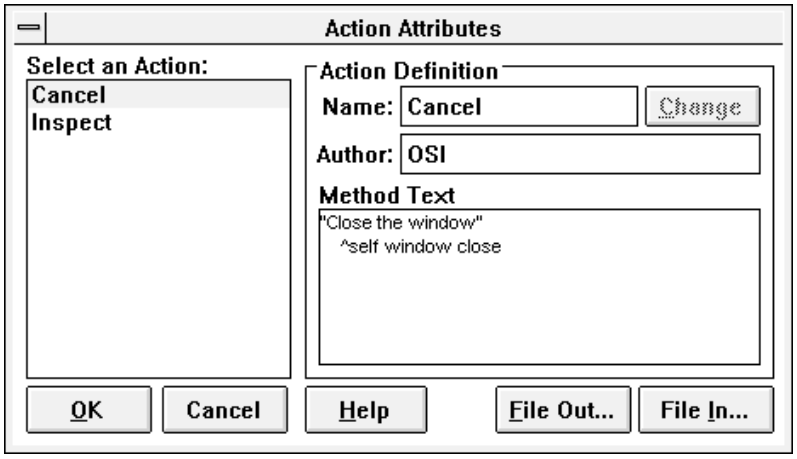
fixedSize
If the button has a bitmap as its label, draw the bitmap at its normal size.

invisible
Makes the button invisible (no label or bitmap). This is useful for creating "sense regions" on top of other visual objects. For this to work properly, exclude the widget from the tab order.

stretchToFit
If the button has a bitmap as its label, stretch or shrink the bitmap to fill the button.

Supported Events **None** (The **clicked** event is directly handled by the widget)

WindowBuilder Extensions



The ActionButton attribute editor allows you to specify the action to attach to the button. The actions are listed in the ListBox on the left. The name of the action, its author, and the Smalltalk code that will be executed are displayed on the right.

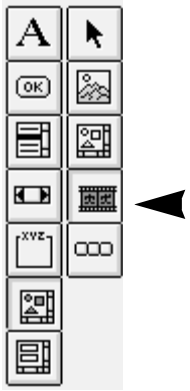
To create a new action, use any of the existing actions as a template. Type in the code that you would like to execute and then change the name of the action (otherwise you will end up modifying the existing action). Notice that the **Change** button turns to **Add** whenever you change the name of the action. The author of the action will always default to be the individual to which the current copy of WindowBuilder Pro is registered.

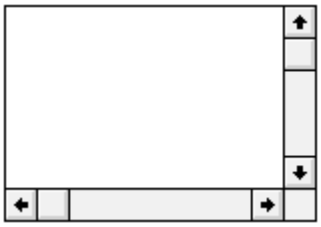
Actions can be created to perform any number of predefined activities. The current window may be referred to as self window. The current button may be referred to as self button. Links to other window may be easily created with code like the following: `MyViewManager new open`. Any compilation errors will be immediately displayed in the text entry field.

The **File Out** button provides a mechanism to file all of the defined actions out to a file. The **File In** button allows you to file in actions from a file.

Holding down the ALT key when invoking the attribute editor will bring up a FileDialog from which a bitmap file may be selected as the label of the button. ActionButtons may have one of three styles: **fixedSize**, **invisible**, or **stretchToFit**. These only apply to buttons that have bitmaps for their labels.

AnimationPane Digitalk





AnimationPanes provide a mechanism to perform simple animations. The widget contains a collection of AnimatedObjects which it manipulates. When the animation is started, a background process is started which sends messages to each active AnimatedObject. For complete details, see Digitalk's documentation or refer to Digitalk's Graphics Demo as an example of how to use them.

Protocol

addObject: *anAnimatedObject*
Add anAnimatedObject to the widget's collection of animated objects.

animate: *anAnimatedObject*
Start *anAnimatedObject* moving continuously.

clear
Stop all animation and remove all objects.

contents: *aCollection*
Set the widget's collection of animated objects

deanimate: *anAnimatedObject*
Stop anAnimatedObject moving continuously.

go
Start animating all active objects.

isActive: *anAnimatedObject*
Answer true if anAnimatedObject is being animated.

makeActive: *anAnimatedObject*
Make anAnimatedObject active, but do not start it animating.

stopAll
Stop all objects from animating.

Supported Events

button1DoubleClick
This event occurs when the user double clicks with the left mouse button within the pane. Use the mouseLocation method to determine the position.

button1Down
This event occurs whenever the left mouse button has been pressed down within the pane.

button1DownShift
This event occurs whenever the left mouse button and shift key have been pressed down within the pane.

button1Move
This event occurs whenever the left mouse button is down and the mouse is moved.

button1Up
This event occurs whenever the left mouse button has been released.

button2DoubleClick
This event occurs when the user double clicks with the right mouse button within the pane.

button2Down
This event occurs whenever the right mouse button has been pressed down within the pane.

button2Move
This event occurs whenever the right mouse button is down and the mouse is moved.

button2Up
This event occurs whenever the right mouse button has been released.

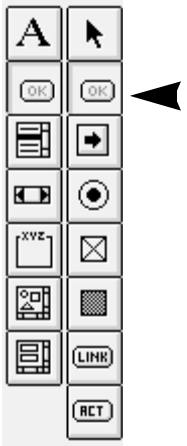
display
This event occurs whenever the widget wants to display itself.

mouseMove
This event occurs whenever the mouse is moved within the pane.

WindowBuilder Extensions

The **noBorders** and **noScrollBars** styles have been added.

Button Digitalk



Buttons provide a mechanism to initiate an action when clicked on. They may be labeled with a text string and will generate a clicked event when pressed. Buttons are commonly used to close the current window, open other windows, and perform actions on data associated with the current window.

Protocol

contents
Return the label of the button.

contents: *aString*
Set the label of the button to the string specified by *aString.*

defaultPushButton
Set the button to be the default push button. Pressing the Enter key is equivalent to clicking this button.

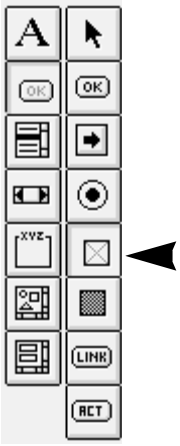
pushButton
Set the Button to be a simple push button.

Supported Events

clicked
This event occurs any time the button is pressed

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

CheckBox Digitalk



CheckBoxes are used to represent a boolean value that is either on or off (true or false). These buttons not only allow the user to set or change a boolean value, but they also act as an indicator of the value's current state.

Protocol

autoCheckBox
Set the automatic style. The widget will manage its own state and automatically toggle itself.

checkBox
Set the simple style. The developer must manage the CheckBox's state when clicked on.

contents
Return the label of the button.

contents: *aString*
Set the label of the button to the string specified by *aString.*

selection
Return the state of the CheckBox - true if it is on, false if off.

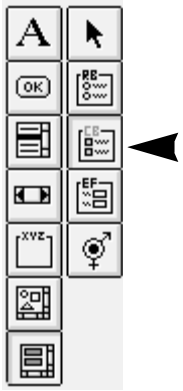
selection: *aBoolean*
Set the state of the CheckBox - true to check it, false to uncheck it.

Supported Events

clicked
This event occurs any time the button is pressed

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

CheckBoxGroup Objectshare



CheckBoxGroup is a special CompositePane subclass that provides a mechanism for the user to select multiple options from a set of options. CheckBoxGroups share the same protocol and may be used interchangeably.

Protocol

contents
Return the collection of labels of the CheckBoxes within the group.

contents: *aCollectionOfStrings*
Set the labels of the CheckBoxes within the group. The number of CheckBoxes is determined by the size of *aCollectionOfStrings.* **Note: This method may only be used before the window is opened.**

label: *aString*
Sets the text label of the GroupBox surrounding the widget.

numColumns: *anInteger*
Set the number of columns in the group.

selectedItems
Answer a collection of the selected items.

selection
Answer the index of the first item selected. The index starts at 1.

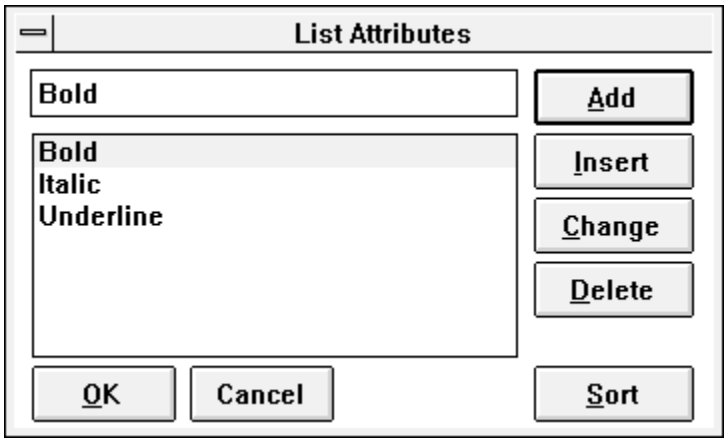
selection: *anObj*
If *anObj* is a collection then select items whose indices are in *anObj*. If *anObj* is an Integer then select the item indexed by *anObj*. Otherwise, select *anObj* in the list.

selections
Answer the indices of the selected items.

Supported Events

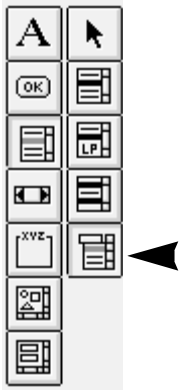
select
This event occurs whenever one of the grouped CheckBoxes is clicked.

WindowBuilder Extensions



CheckBoxGroups share the same attribute editor with ListBoxes, ComboBoxes, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted.

ComboBox Digitalk



ComboBoxes are hybrids between EntryFields and ListBoxes. It consists of a text entry field which is always visible and a list which may be displayed all of the time or only when the user selects the control.

ComboBoxes come in three varieties: simple, dropDown, and dropDownList. Simple ComboBoxes always display their lists. A dropDown ComboBox has a button next to the entry field which when clicked causes the list to appear. A dropDownList ComboBox only allows text entries that are items in the list.

Protocol

contents
Return the contents of the ComboBox.

contents: *aCollection*
Set the contents of the ComboBox to *aCollection.*

indexOf: *aString*
Answer the index of the item *aString* in the list.

insertItem: *aString*
Insert *aString* into the list.

deleteAll
Delete the entire list

dropDown
Set the style of the ComboBox to **dropDown.**

dropDownList
Set the style of the ComboBox to **dropDownList.**

selectedItem
Answer the item selected in the ListBox

selection
Answer the index of the selected item. The index starts at 1.

selection: *anObject*
Select the item indicated by *anObject. anObject* is either an index into the list or a string with which to search the list.

simpleList
Set the style of the ComboBox to simple.

text
Answer the text in the ComboBox's text entry field.

text: *aString*
Set the text of the ComboBox to aString.

Supported Events

charInput
This event occurs whenever any character is typed.

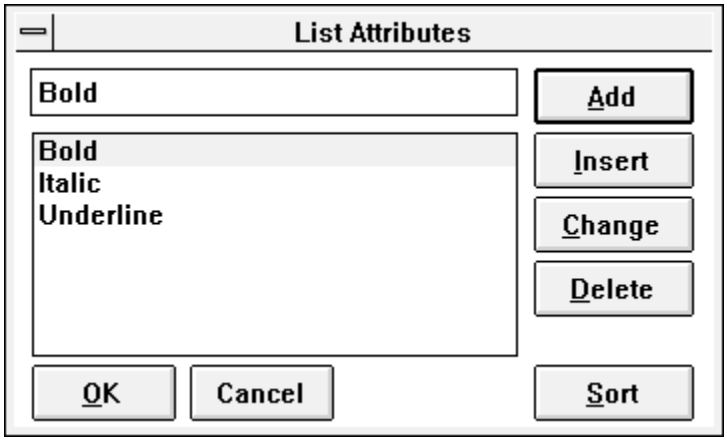
getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

listVisible
This happens when the user clicks the ComboBox button to pull down the list.

select
This event occurs whenever the user selects an item from the list component of the ComboBox.

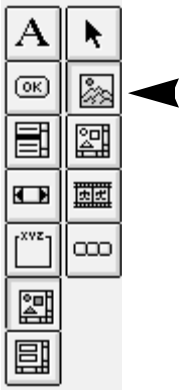
textChanged
This event occurs any time the text of the entry field changes. This event is sent when the contents of the ComboBox is set, the user types into the entry field, or the user selects from the list.

WindowBuilder Extensions



ComboBoxes share the same attribute editor with ListBoxes, RadioButtonGroups, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted.

CPStaticGraphic Objectshare



CPStaticGraphic is a replacement for Digitalk's StaticGraphic widget. It allows you to display a bitmap image within a window.

Protocol

cpContents
Return the name of the file in which its bitmap is stored.

cpContents: *aString*
Set the contents of the graphic. If *aString* is a valid bitmap file name (.BMP), the widget will display the bitmap.

contents
Return the bitmap displayed by the widget.

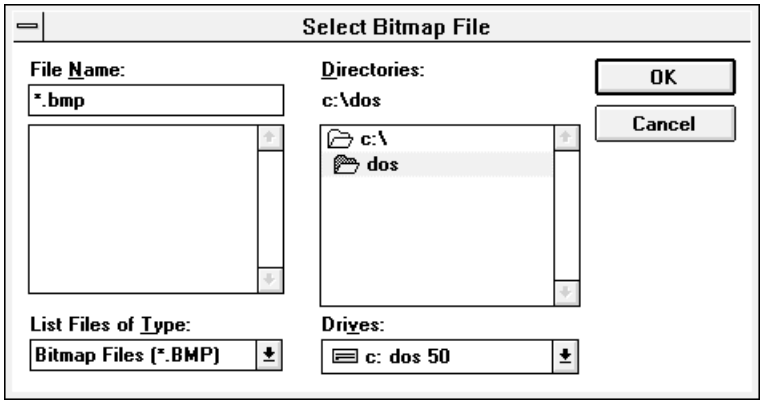
contents: *aBitmap*
Set the contents of the widget to the bitmap specified by *aBitmap.*

fixedSize
If the widget has a bitmap as its label, draw the bitmap at its normal size.

stretchToFit
If the widget has a bitmap as its label, stretch or shrink the bitmap to fill the widget.

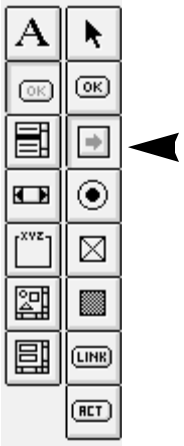
Supported Events **None**

WindowBuilder Extensions



The attribute editor for CPStaticGraphics allows you to select a bitmap file (.BMP) from disk.

DrawnButton Digitalk



DrawnButtons act exactly like regular push buttons. They are labeled with graphic images rather than text strings. We have also provided an invisible style which can be used to create “sense regions” on other graphic images. For example, you could place invisible buttons on top of a large imported bitmap.

Protocol

cpContents
Return the label of the button or the name of the file in which its bitmap is stored.

cpContents: *aString*
Set the contents of the button. If *aString* is a valid bitmap file name (.BMP), the button will display the bitmap. If not, the button will look like a simple Button with *aString* as its label.

contents
Return the bitmap displayed by the button

contents: *aBitmap*
Set the contents of the button to the bitmap specified by *aBitmap.*

fixedSize
If the widget has a bitmap as its label, draw the bitmap at its normal size.

invisible
Makes the button invisible (no label or bitmap). This is useful for creating “sense regions” on top of other visual objects. For this to work properly, exclude the widget from the tab order.

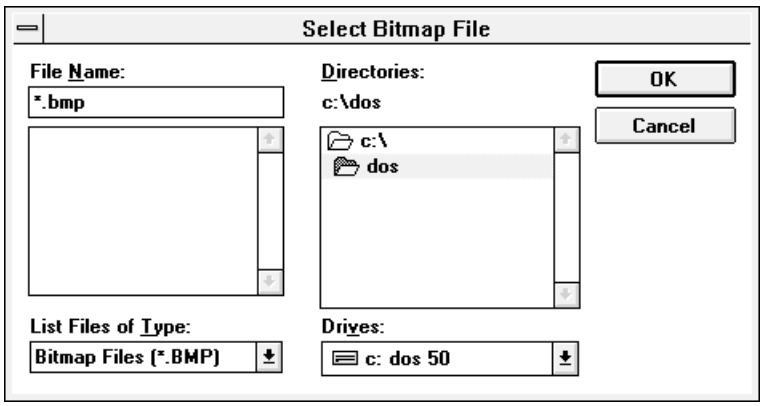
stretchToFit
If the widget has a bitmap as its label, stretch or shrink the bitmap to fill the widget.

Supported Events

clicked
This event occurs any time the button is pressed

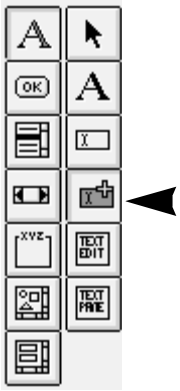
getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

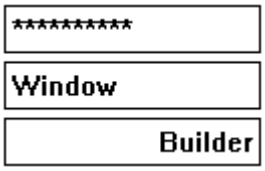
WindowBuilder Extensions



The attribute editor for DrawnButtons allows you to select a bitmap file (.BMP) from disk.

EnhancedEntryField Objectshare





EnhancedEntryFields are advanced single line entry fields that provide character and field level validation. Character level validations include: Alpha, AlphaNumeric, Boolean, Integer, and PositiveInteger. Field level validations include: Date, PhoneNumberUS, SSN, ZipCodeUS, etc.

In addition to validation functions, special behaviors may be specified for gaining and losing focus. Additional styles such as **readonly** and **password** are available. The maximum number of allowable characters may be set.

Under OS/2, justification (right, left, centered) and auto-tabbing may be specified.

Protocol

autoTab
Set the entry field to automatically tab to the next field when the maximum number of characters has been entered. Currently not supported under Windows.

case: *aSymbol*
Specify any automatic case conversion that should take place when the field looses focus. Allowable values are: #**UPPER**, #**lower**, #**Proper**, and #**Unchanged**.

centered
Specify center justification. Currently not supported under Windows.

character: *aSymbol*
Set the character level validation function.

contents
Return the contents of the field as a string.

contents: *aString*
Set the contents of the field adjusting the case as required.

field: *aSymbol*
Set the field level validation function.

getFocus: *aSymbol*
Specify where the cursor should be placed when the field gets focus. Allowable values include: #**selectAll**, #**selectFirst**, and #**selectLast**.

getSelection
Returns the starting and ending character positions of the current selection of the field.

left
Specify left justification. This is the default.

maxSize: *anInteger*
Set the maximum number of characters allowed in the field.

password
Make the field password protected.

readonly
Make the field read only.

right
Specify right justification. Currently not supported under Windows.

selectAll
Select all of the text in the field.

ok...
These are the field level validation methods. In addition to validating the field, they may also reformat the field. If the field is invalid, an error message will be presented to the user.

ok...: *aChar*
These are the character level validation methods. If the character is not valid, it will not be entered into the field.

Supported Events

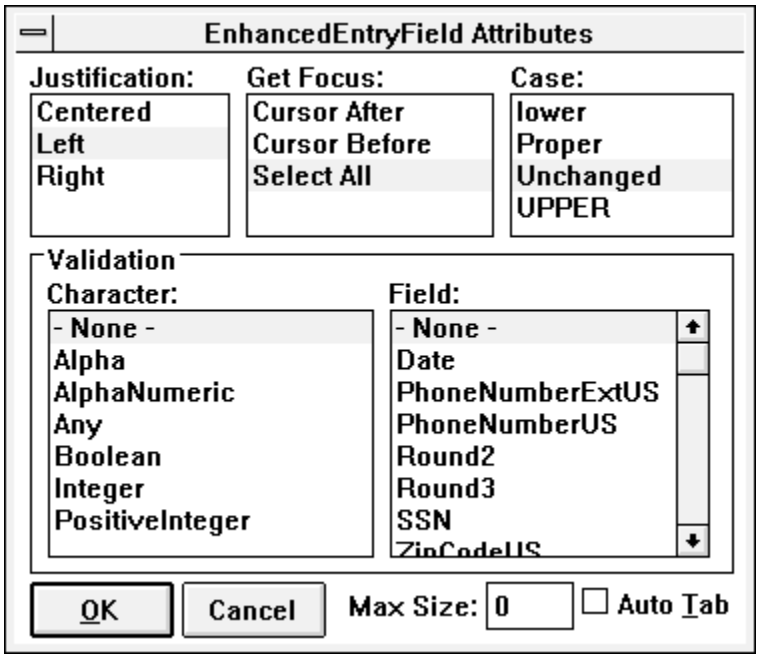
getContents
This event occurs when the field first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

gettingFocus
This event occurs when the field gets the input focus either by the user clicking in the field or tabbing to it.

losingFocus
This event occurs when the field loses the input focus either by the user clicking outside the field or tabbing from it.

textChanged
This event occurs whenever a character is typed within the field.

WindowBuilder Extensions



The EnhancedEntryField attribute editor allows you to set the justification (OS/2 only), get focus cursor position, case adjustment on losing focus, maximum number of characters, and auto-tabbing (OS/2 only).

Character and Field level validation may also be specified. The character level validations provided are:

- **Alpha**
Only the characters $A-$Z, $a-$z, and space are allowed.

- **AlphaNoSpace**
Only the characters $A-$Z and $a-$z are allowed.

- **AlphaNumeric**
Only alpha characters, the digits $0-$9, and space are allowed.

- **AlphaNumericNoSpace**
Only alpha characters and the digits $0-$9 are allowed.

- **Any**
Any character is allowed

- **Boolean**
Only the characters $T, $t, $F, $f, $Y, $y, $N, and $n are allowed.

- **Integer**
Only acceptable integers (positive or negative) are allowed.

- **Numeric**
Only acceptable numbers (positive or negative) are allowed.

- **PositiveInteger**
Only the digits $0-$9 are allowed.

The field level validations provided are:

- **Date**
The contents of the field must be a valid date. A variety of formats are supported. The field is automatically reformatted to reflect the system date format.

- **PhoneNumberExtUS**
The contents of the entry field are reformatted as a standard US phone number complete with extension. Any Alpha characters will be converted to there phone number equivalents.

- **PhoneNumberUS**
The contents of the entry field are reformatted as a standard US phone number. An error is generated if the field contains other than seven or ten characters. Any Alpha characters will be converted to there phone number equivalents.

- **Round2**
Round the contents of the field to two decimal places.

- **Round3**
Round the contents of the field to three decimal places.

- **SSN**
The contents must be a valid Social Security Number. The contents will be automatically reformatted.

- **ZipCodeUS**
The contents must be a valid five or nine character US zip code. The contents will be reformatted if required.

Additional validation functions may be added easily. All of the validation functions are public methods of the EnhancedEntryField class and begin with the characters ‘ok’. Character level validations take one argument - the character itself. The routine should respond with true or false depending on whether the character is valid. For example, to create a validation function that would only allow the asterisk character to be entered, the following code would be required:

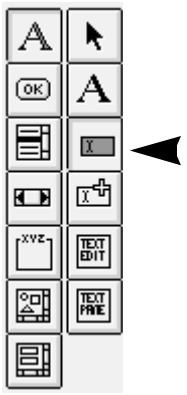
```
okAsteriskOnly: aChar
   ^aChar == $*
```

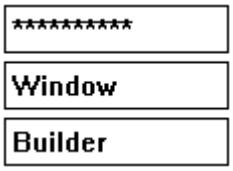
Field level validations take no arguments and work on the entire contents of the field. They should return true or false depending on whether the field in valid. They may also optionally reformat the field as required. A simple field level validation that would test whether the contents is a palindrome could be coded as follows:

```
okPalindrome
   ^self contents =
   self contents reversed
```

The attribute editor automatically displays all validation functions that it identifies within the EnhancedEntryField class.

EntryField Digitalk





EntryFields are used to provide single line text entry and edit capabilities. If the user types more text than can be accommodated within the field, it will automatically scroll. Additional styles such as **readonly** and **password** are available. The **readonly** style places the field in output-only mode. Users can select and copy text within it but they can not change the text. The **password** style causes any entered values to be replaced by a series of asterisks ("*"). The contents of the field may be pre-set by entering text into WindowBuilder's text/label field.

Protocol

contents
Return the contents of the field as a string.

contents: *aString*
Set the contents of the field to the string *aString.*

getSelection
Returns the starting and ending character positions of the current selection of the field.

maxSize: *anInteger*
Set the maximum number of characters allowed in the field.

password
Make the field password protected.

readonly
Make the field read only.

selectAll
Select all of the text in the field.

Supported Events

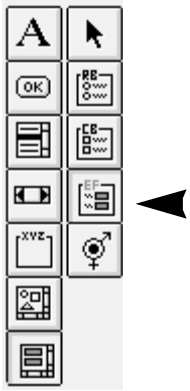
getContents
This event occurs when the widget first comes up. This is a good opportunity to set the field's contents using the **contents:** method.

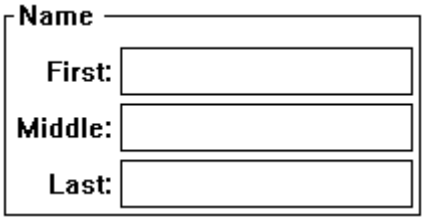
textChanged
This event occurs whenever a character is typed within the field.

WindowBuilder Extensions

WindowBuilder adds the **readonly** and **password** styles.

EntryFieldGroup Objectshare





EntryFieldGroup is a special CompositePane subclass that provides a mechanism to rapidly create data entry forms. The widget may be displayed with or without a GroupBox or may optionally have a vertical scroll bar.

Protocol

contents
Returns a dictionary that represents the contents of the widget. The keys of the dictionary are the StaticText labels. The value are the contents of the corresponding EntryFields.

contents: *aDictionaryOfStrings*
The dictionary's keys set the labels of the StaticText labels within the group. The dictionary's values provide the contents of the EntryFields. If *aDictionaryOfStrings* is some other type of collection than a dictionary, the items of the collection will be used as the StaticText labels and the EntryFields will be initialized to empty strings. The number of EntryFields is determined by the size of *aDictionaryOfStrings.* **Note: This method may only be used before the window is opened.**

fieldClass
Returns the EntryField subclass to be used within the widget. EntryFieldGroup can easily be subclassed to use a different text entry class (such as the EnhancedEntryField).

label: *aString*
Sets the text label of the GroupBox surrounding the widget.

noGroupBox
Sets the style of the widget to not include a surrounding GroupBox.

setLabelFont: *aFont*
Sets the font of the StaticText labels to *aFont.*

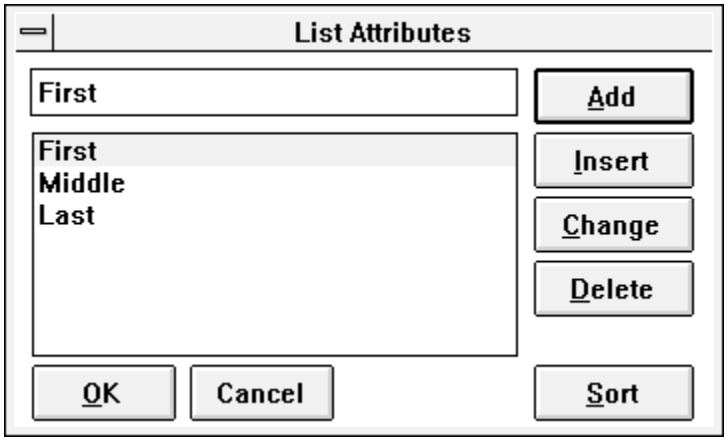
setValueFont: *aFont*
Sets the font of the EntryFields to *aFont.*

verticalScrollBar
Sets the style of the widget to have a vertical scroll bar.

Supported Events

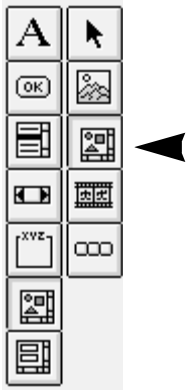
textChanged
This event occurs whenever the text of any one of the EntryFields is changed

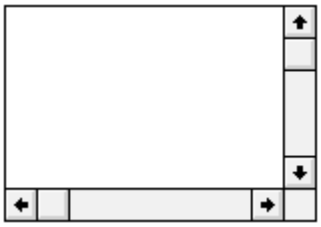
WindowBuilder Extensions



EntryFieldGroups share the same attribute editor with ListBoxes, ComboBoxes, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted.

GraphPane Digitalk





GraphPanes are very powerful subpanes that provide extensive support for visual display and user input. Using a GraphPane's pen is the easiest way to add custom graphics to an application. By associating a message with any of its mouse or keyboard events, a programmer can handle simple user interaction. For complete details, see Digitalk's documentation.

Protocol

erase
Clear the contents of the pane.

mouseLocation
Return the mouse location as of the last mouse input event.

noBorders
Set the style to not include a border.

noScrollBars
Set the style not to include any scroll bars.

stretch: *anInteger*
Set the scaling attributes of the pane. 0 means no stretch. 1 means stretch while maintaining the aspect ration. Anything else means stretch according to the window's dimensions.

Supported Events

button1DoubleClick
This event occurs when the user double clicks with the left mouse button within the pane. Use the mouseLocation method to determine the position.

button1Down
This event occurs whenever the left mouse button has been pressed down within the pane.

button1DownShift
This event occurs whenever the left mouse button and shift key have been pressed down within the pane.

button1Move
This event occurs whenever the left mouse button is down and the mouse is moved.

button1Up
This event occurs whenever the left mouse button has been released.

button2DoubleClick
This event occurs when the user double clicks with the right mouse button within the pane.

button2Down
This event occurs whenever the right mouse button has been pressed down within the pane.

button2Move
This event occurs whenever the right mouse button is down and the mouse is moved.

button2Up
This event occurs whenever the right mouse button has been released.

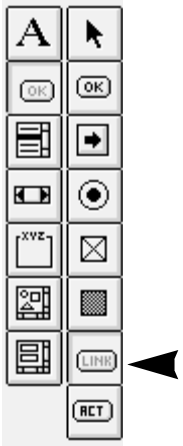
display
This event occurs whenever the widget wants to display itself.

mouseMove
This event occurs whenever the mouse is moved within the pane.

WindowBuilder Extensions

The **noBorders** and **noScrollBars** styles have been added.

LinkButton Objectshare



ActionButtons provide a mechanism to rapidly link windows together without writing Smalltalk code. This gives the developer a quick and easy way to prototype screen flow with very little effort. In addition to specifying the ViewManager subclass that should be linked to, you may also specify the type of link. Independent links open windows that are logically independent of the windows that created them. Child links cause the new windows to become logical children of the windows that create them. Child windows float on top of their parent, minimize with them, and close when they close. Sibling links are a variation on Child links where the new window becomes a child of the parent of the window that created it.

Since LinkButtons inherit from DrawnButton, they may take on the visual appearance of either a simple Button or may have a graphic attached to them. We have also provided an invisible style which can be used to create "sense regions" on other graphic images. For example, you could place invisible buttons on top of a large imported bitmap.

Protocol

cpContents
Return the label of the button or the name of the file in which its bitmap is stored.

cpContents: *aString*
Set the contents of the button. If *aString* is a valid bitmap file name (.BMP), the button will display the bitmap. If not, the button will look like a simple Button with *aString* as its label.

contents
Return the bitmap displayed by the button.

contents: *aBitmap*
Set the contents of the button to the bitmap specified by *aBitmap.*

fixedSize
If the button has a bitmap as its label, draw the bitmap at its normal size.

invisible
Makes the button invisible (no label or bitmap). This is useful for creating "sense regions" on top of other visual objects. For this to work properly, exclude the widget from the tab order.

link: *aSymbol*
Specify the ViewManager subclass to which the button should link. The argument should be specified as a Symbol.

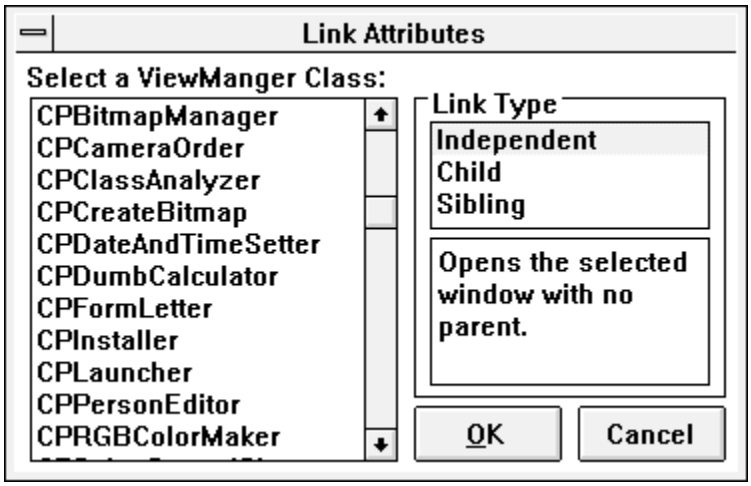
stretchToFit
If the button has a bitmap as its label, stretch or shrink the bitmap to fill the button.

type: *aSymbol*
Specify the type of link that should be performed. The allowable choices are: #Independent, #Child, and #Sibling.

Supported Events

None (The **clicked** event is directly handled by the widget)

WindowBuilder Extensions

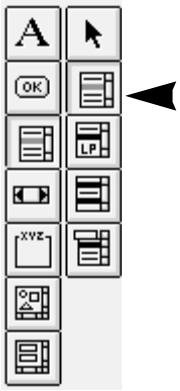


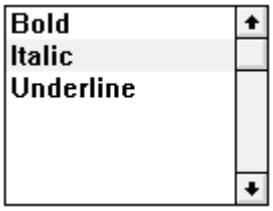
The LinkButton attribute editor allows you to specify the ViewManager subclass that the button should link to. The legal ViewManager subclasses appear in the ListBox on the left (the list varies based on the windows defined in the system – the list that you see will likely be different). In order for the LinkButton to automate a link, the target ViewManager subclass must define either an **createViews** or an **open** method and have been created with WindowBuilder (e.g., respond "true" to the `wbCreated` message).

On the right side of the editor, the type of link may be specified. All legal ViewManagers may be linked via **Independent** links. This type of link results in no logical relationship between the windows. If a ViewManager has a **createView** method, other link options are available (Note that if a ViewManager only defines an **open** method, this will have to be converted to a **createView** method to use these additional links). **Child** links establish a parent/child relationship between the new window and the one that created it. The new child window will float on top of its parent, minimize with it, and close with it. **Sibling** links should be used in the special case that a child window wants to open another child for its parent. This would often be the case if you had a "Desktop" window that acted as the parent of all of your application windows.

Holding down the ALT key when invoking the attribute editor will bring up a FileDialog from which a bitmap file may be selected as the label of the button. LinkButtons may have one of three styles: **fixedSize**, **invisible**, or **stretchToFit**. These only apply to buttons that have bitmaps for their labels.

ListBox Digitalk





ListBoxes provide a general selection capability. The user is presented with a list of choices (either strings or bitmaps) and may select one with the mouse.

Protocol

contents
Return the list.

contents: *aCollection*
Set the contents of the widget to *aCollection.*

deleteAll
Delete the entire list.

draw: *aBitmap* **for:** *aString*
Use *aBitmap* in place of *aString* when displaying the contents of the list.

indexOf: *aString*
Return the index of *aString* in the list.

insertItem: *aString*
Insert the item *aString* into the list.

ownerDrawFixed
Set the style of the widget to the fixed height owner draw style.

ownerDrawVariable
Set the style of the widget to the variable height owner draw style.

selectedItem
Return the item currently selected in the widget.

selection
Return the index of the item currently selected in the widget.

selection: *anObject*
Select the line indicated by *anObject. anObject* may either be an index into the list or a string contained in the list.

Supported Events

charInput
This event occurs whenever a character is typed in the widget.

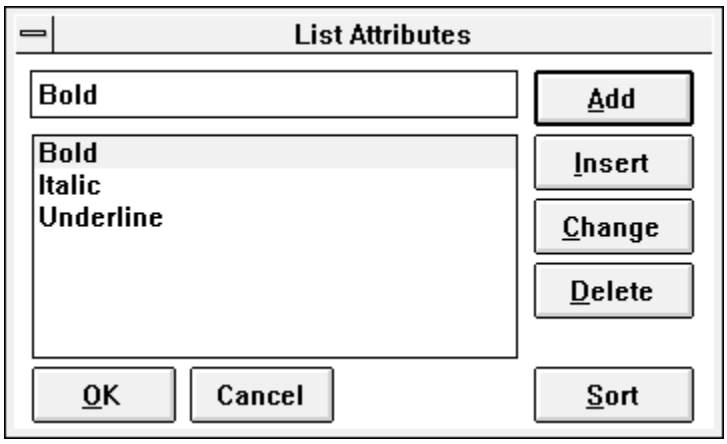
doubleClickSelect
This event occurs whenever an item is double clicked on with the mouse.

drawItem
This event occurs whenever a user drawn item is included in the list. This is rarely used.

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

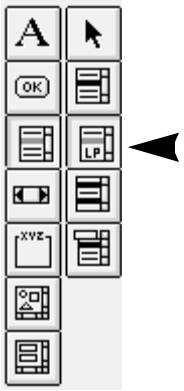
select
This event occurs whenever an unselected item is selected. Items may be selected with the mouse or keyboard.

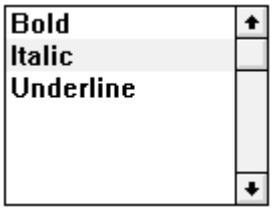
WindowBuilder Extensions



ListBoxes share the same attribute editor with ComboBoxes, RadioButtonGroups, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted

ListPane Digitalk





ListPanes provide a general selection capability and are functionally equivalent to ListBoxes. The user is presented with a list of choices and may select one with the mouse. In some Smalltalk/V platforms, ListPanes are implemented entirely in Smalltalk as opposed to ListBoxes which are real operating system widgets.

Protocol

contents
Return the list.

contents: *aCollection*
Set the contents of the widget to *aCollection.*

deleteAll
Delete the entire list.

draw: *aBitmap* **for:** *aString*
Use *aBitmap* in place of *aString* when displaying the contents of the list.

indexOf: *aString*
Return the index of *aString* in the list.

insertItem: *aString*
Insert the item *aString* into the list.

ownerDrawFixed
Set the style of the widget to the fixed height owner draw style.

ownerDrawVariable
Set the style of the widget to the variable height owner draw style.

selectedItem
Return the item currently selected in the widget.

selection
Return the index of the item currently selected in the widget.

selection: *anObject*
Select the line indicated by *anObject. anObject* may either be an index into the list or a string contained in the list.

Supported Events

charInput
This event occurs whenever a character is typed in the widget.

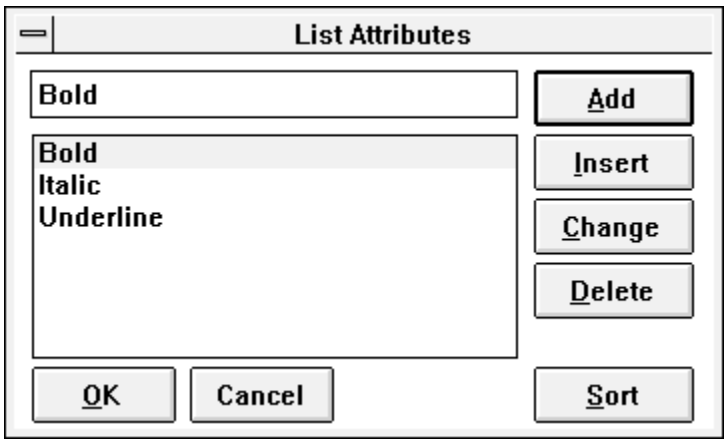
doubleClickSelect
This event occurs whenever an item is double clicked on with the mouse.

drawItem
This event occurs whenever a user drawn item is included in the list. This is rarely used.

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

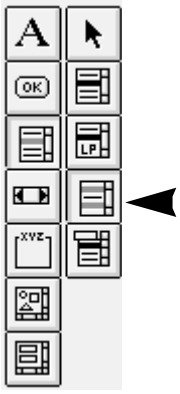
select
This event occurs whenever an unselected item is selected. Items may be selected with the mouse or keyboard.

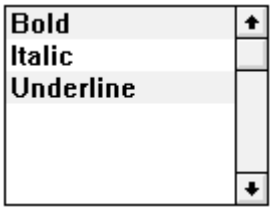
WindowBuilder Extensions



ListPanes share the same attribute editor with ComboBoxes, RadioButtonGroups, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted

MultipleSelectListBox Digitalk





MultipleSelectionListBoxes are identical to single selection ListBoxes with the added ability to select multiple choices from the list.

Protocol

contents
Return the list.

contents: *aCollection*
Set the contents of the widget to *aCollection.*

deleteAll
Delete the entire list.

draw: *aBitmap* **for:** *aString*
Use *aBitmap* in place of *aString* when displaying the contents of the list.

indexOf: *aString*
Return the index of *aString* in the list.

insertItem: *aString*
Insert the item *aString* into the list.

ownerDrawFixed
Set the style of the widget to the fixed height owner draw style.

ownerDrawVariable
Set the style of the widget to the variable height owner draw style.

selectedItems
Return the collection of items currently selected in the widget.

selection
Return the index of the first item selected in the widget.

selection: *anObject*
Select the lines indicated by *anObject. anObject* may either be a collection of indices, an index into the list or a string contained in the list.

selections
Return a collection of the indices of each of the items selected in the widget.

Supported Events

charInput
This event occurs whenever a character is typed in the widget.

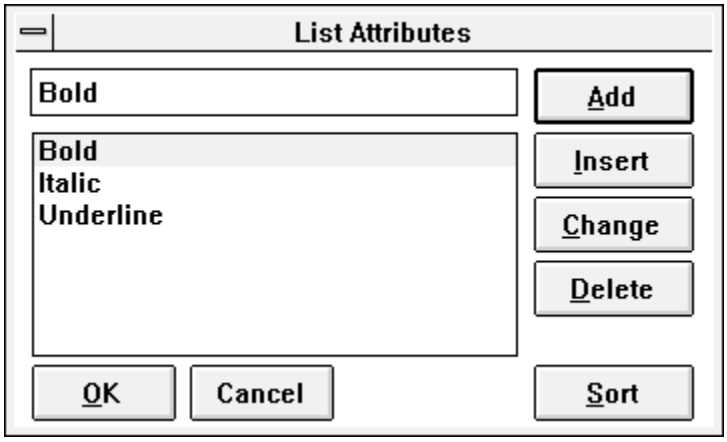
doubleClickSelect
This event occurs whenever an item is double clicked on with the mouse.

drawItem
This event occurs whenever a user drawn item is included in the list. This is rarely used.

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

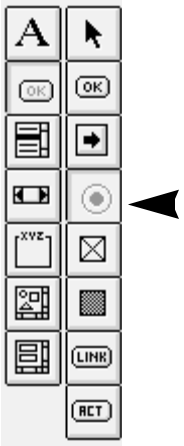
select
This event occurs whenever an unselected item is selected. Items may be selected with the mouse or keyboard.

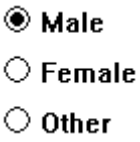
WindowBuilder Extensions



MultipleSelectionListBoxes share the same attribute editor with ComboBoxes, RadioButtonGroups, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted.

RadioButton Digitalk





RadioButtons are used to represent mutually exclusive choices. They look and act like the radio buttons found in cars. Only one button in a group may be selected (indicated by having a dot in the center).

Protocol

autoRadioButton
Set the automatic style. The widget will manage its own state and automatically toggle itself.

contents
Set the label of the button.

contents: *aString*
Set the label of the button to the string specified by *aString.*

radioButton
Set the simple style. The developer must manage the RadioButton's state when clicked on.

selection
Return the state of the RadioButton - true if it is on, false if off.

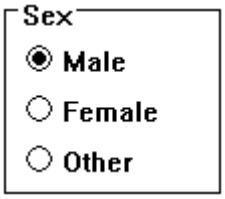
selection: *aBoolean*
Set the state of the RadioButton - true to check it, false to uncheck it.

Supported Events

clicked
This event occurs any time the button is pressed

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

RadioButtonGroup Objectshare



RadioButtonGroup is a special CompositePane subclass that provides a mechanism for the user to select one mutually exclusive option from a set of options. RadioButtonGroups share the same protocol as ListBoxes and may be used interchangeably.

Protocol

contents
Return the collection of labels of the RadioButtons within the group.

contents: *aCollectionOfStrings*
Set the labels of the RadioButtons within the group. The number of CheckBoxes is determined by the size of *aCollectionOfStrings.* **Note: This method may only be used before the window is opened.**

label: *aString*
Sets the text label of the GroupBox surrounding the widget.

numColumns: *anInteger*
Set the number of columns in the group.

selectedItem
Return the item currently selected in the widget.

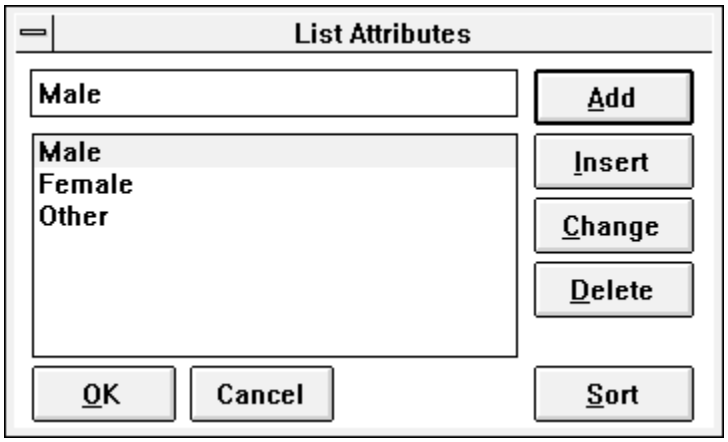
selection
Return the index of the item currently selected in the widget.

selection: *anObject*
Select the line indicated by *anObject. anObject* may either be an index into the list or a string contained in the list.

Supported Events

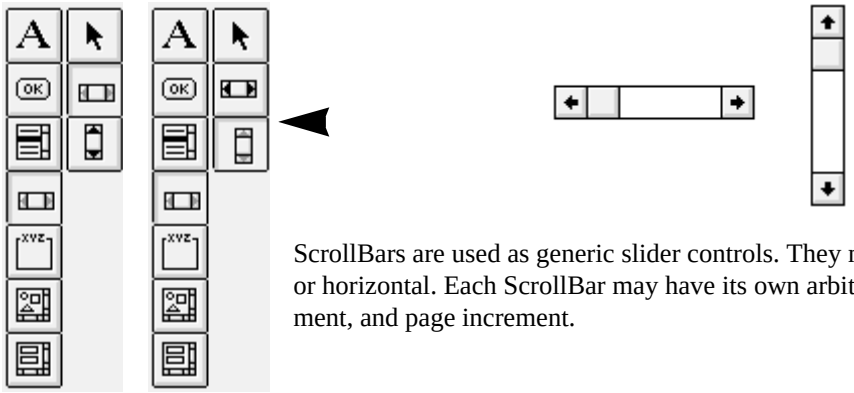
select
This event occurs whenever one of the grouped RadioButtons is clicked.

WindowBuilder Extensions



RadioButtonGroups share the same attribute editor with ListBoxes, ComboBoxes, and other list oriented widgets. The list attribute editor provides a means to specify a list of items. Items may be added, inserted, changed, deleted, and sorted.

ScrollBar Digitalk



ScrollBars are used as generic slider controls. They may be either vertical or horizontal. Each ScrollBar may have its own arbitrary range, line increment, and page increment.

Protocol

horizontal
Set the horizontal style.

lineIncrement: *anInteger*
Set the line increment to be *anInteger*.

maximum: *anInteger*
Set the maximum value the ScrollBar can have.

minimum: *anInteger*
Set the minimum value the ScrollBar can have.

pageIncrement: *anInteger*
Set the page increment to be *anInteger*.

position
Return the position of the slider portion of the ScrollBar.

position: *anInteger*
Set the position of the slider portion of the ScrollBar.

vertical
Set the vertical style.

Supported Events

end
This event occurs when slider is dragged to the bottom or right end.

endScroll
This event occurs when the user finishes scrolling.

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's position using the **position:** method.

home
This event occurs when slider is dragged to the top or left end.

nextLine
This event occurs any time the down or right arrow is clicked.

nextPage
This event occurs when the mouse is clicked in the area between the slider and the next arrow button.

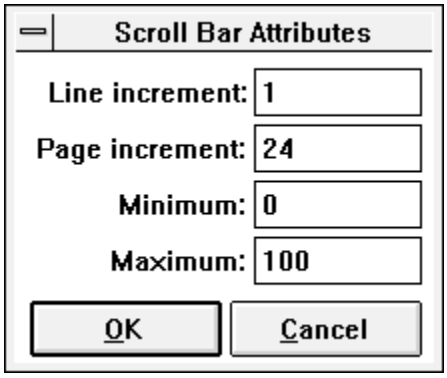
prevLine
This event occurs any time the up or left arrow is clicked.

prevPage
This event occurs when the mouse is clicked in the area between the slider and the previous arrow button.

sliderTrack
This event occurs as the user drags the slider.

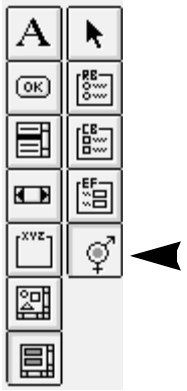
sliderPosition
This event occurs when the user finishes dragging the slider.

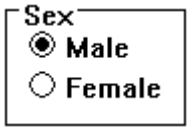
WindowBuilder Extensions



The ScrollBar attribute editor allows you to specify the line and page increments to associate with the ScrollBar. The minimum and maximum allowed values may also be specified.

SexPane Objectshare





SexPane is a special-purpose CompositePane subclass used to specify the sex of an individual. Rather that responding to individual events of its components, one need only respond to SexPane's higher-order protocol.

Protocol

contents
Return the sex of the widget. Allowable values are #male and #female.

contents: *aSymbol*
Set the sex of the widget. Allowable values are #male and #female.

sex
Return the sex of the widget. Allowable values are #male and #female.

sex: *aSymbol*
Set the sex of the widget. Allowable values are #male and #female.

Supported Events

setToFemale
This event occurs whenever the female RadioButton is clicked.

setToMale
This event occurs whenever the male RadioButton is clicked.

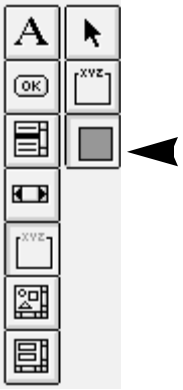
sexChanged
This event occurs whenever the sex of the widget changes.

WindowBuilder Extensions

As with all normal CompositePane classes, the attribute editor for the SexPane widget is another copy of WindowBuilder Pro. Editing the definition of SexPane class will change every instance of it.

StaticBox

Digitalk



The StaticBox class is used to draw filled and outline boxes. They may either be black, gray, or white.

Protocol

blackFrame
Set the static black frame style.

blackRectangle
Set the static black rectangle style.

grayFrame
Set the static gray frame style.

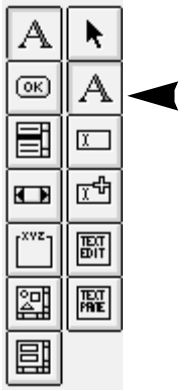
grayRectangle
Set the static gray rectangle style.

whiteFrame
Set the static white frame style.

whiteRectangle
Set the static white rectangle style.

Supported Events **None**

StaticText Digitalk



Left Justified

Centered

Right Justified

The StaticText class provides a simple text output capability. StaticText widgets are used as labels within forms. They are often used in conjunction with EntryFields.

Protocol

centered
Set the center justified style.

contents
Return the contents of the widget.

contents: *aString*
Set the contents of the widget.

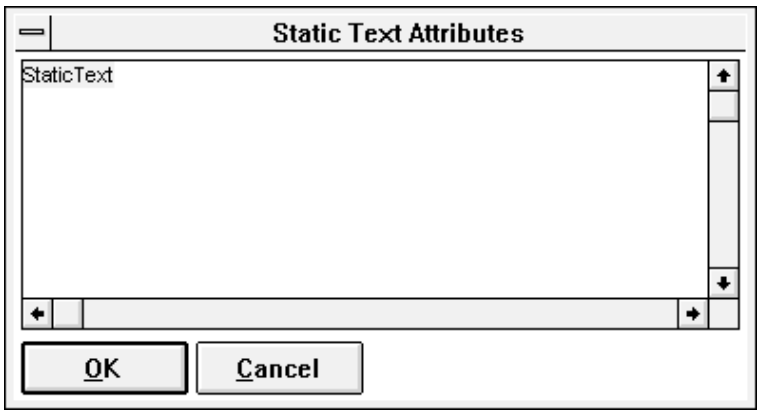
leftJustified
Set the left justified style.

rightJustified
Set the right justified style.

Supported Events

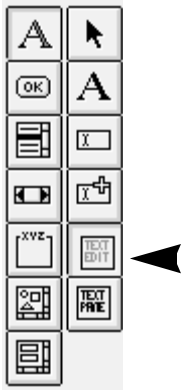
getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

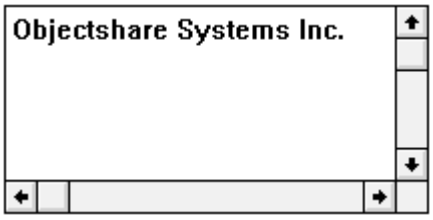
WindowBuilder Extensions



The StaticText attribute editor allows you enter multi-line text for the pane.

TextEdit Digitalk





TextEdits are used to provide multi-line text entry and edit capabilities. If the user types more text than can be accommodated within the field, it will automatically scroll.

Protocol

contents
Return the contents of the field as a string.

contents: *aString*
Set the contents of the field to the string *aString.*

getSelection
Returns the starting and ending character positions of the current selection of the field.

maxSize: *anInteger*
Set the maximum number of characters allowed in the field.

noBorders
Set the style to not include a border.

noScrollBars
Set the style to not include scroll bars (the default).

scrollBars
Set the style to include scroll bars.

selectAll
Select all of the text in the field.

wordWrap
Set the style to include a vertical scrollbar and automatically word wrap.

Supported Events

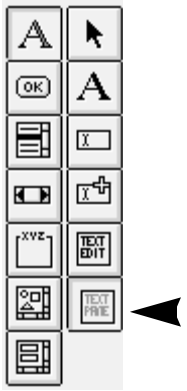
getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

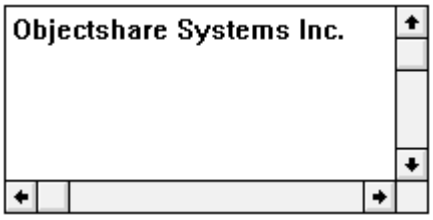
textChanged
This event occurs whenever a character is typed within the field.

WindowBuilder Extensions

WindowBuilder adds the **noBorders**, **scrollBars**, and **wordWrap** styles.

TextPane Digitalk





TextPanes are used to provide multi-line text entry and edit capabilities. If the user types more text than can be accommodated within the field, it will automatically scroll. In some Smalltalk/V platforms, TextPanes are implemented entirely in Smalltalk as opposed to TextBoxes which are real operating system widgets.

Protocol

contents
Return the contents of the field as a string.

contents: *aString*
Set the contents of the field to the string *aString.*

cr
Append a line-feed to the text in the field.

nextPut: *aCharacter*
Add *aCharacter* to the end of the text.

nextPutAll: *aString*
Add *aString* to the end of the text.

noBorders
Set the style to not include a border.

noScrollBars
Set the style to not include scroll bars (the default).

selectAfter: *aPoint*
Place the selection after *aPoint.*

selectAll
Select all of the text in the field.

selectAtEnd
Place the selection at the end of the text.

selectBefore: *aPoint*
Place the selection before *aPoint.*

selectedItem
Return the currently selected text.

selectFrom: *start* **to:** *end*
Select the rectangle with origin *start* and extent *end.*

Supported Events

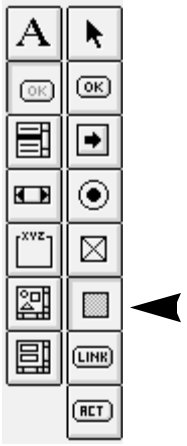
getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

save
This event occurs whenever the text of the pane is saved.

WindowBuilder Extensions

WindowBuilder adds the **noBorders** and **scrollBars** styles.

ThreeStateButton Digitalk



ThreeStateButtons are almost exactly like CheckBoxes except that they may be in an indeterminate state (neither checked or unchecked). The widget will return either **true** or **false** if it is checked or unchecked and will return **nil** if in an indeterminate state. These buttons not only allow the user to set or change their values, but they also act as an indicator of the value's current state.

ThreeStateButtons are often used in the case that a set of selected objects do not share the same value of some characteristic.

Protocol

autoThreeState
Set the automatic style. The widget will manage its own state and automatically toggle itself.

contents
Return the label of the button.

contents: *aString*
Set the label of the button to the string specified by *aString.*

selection
Return the state of the CheckBox - true if it is on, false if off.

selection: *aBooleanOrNil*
Set the state of the widget - true to check it, false to uncheck it, nil to become indeterminate.

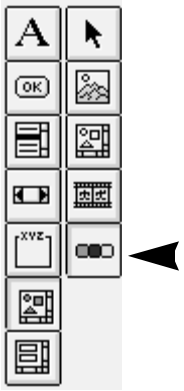
threeState
Set the simple style. The developer must manage the widget's state when clicked on.

Supported Events

clicked
This event occurs any time the button is pressed

getContents
This event occurs when the widget first comes up. This is a good opportunity to set the widget's contents using the **contents:** method.

WBToolBar Objectshare



WBToolBar provides a general purpose horizontal tool bar control similar to that found in dozens of commercial applications. Each button has a bitmap that is stored in the CPBitmapDict and may optionally specify selectors for the left and right mouse buttons as well as the spacing between it and its neighbor.

Protocol

add: *aSelector*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *aSelector*. Its left button selector is *aSelector*. It will not have a right button selector.

add: *aSelector* **rbSelector:** ***rbSelector***
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *aSelector*. Its left button selector is *aSelector*. Its right button selector is *rbSelector*.

add: *bitmapSelector* **selector:** *aSelector*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *bitmapSelector*. Its left button selector is *aSelector*. It will not have a right button selector.

add: *bitmapSelector* **selector:** *aSelector* **rbSelector:** *rbSelector*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *bitmapSelector*. Its left button selector is *aSelector*. Its right button selector is *rbSelector*.

add: *bitmapSelector* **selector:** *aSelector* **spaces:** *numSpaces*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *bitmapSelector*. Its left button selector is *aSelector*. It will be *numSpaces* from its neighbor.

add: *bitmapSelector* **selector:** *aSelector* **spaces:** *numSpaces* **rbSelector:** *rbSelector*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *bitmapSelector*. Its left button selector is *aSelector*. It will be *numSpaces* from its neighbor and its right button selector is *rbSelector*.

add: *aSelector* **spaces:** *numSpaces*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *aSelector*. Its left button selector is *aSelector*. It will be *numSpaces* from its neighbor.

add: *aSelector* **spaces:** *numSpaces* **rbSelector:** *rbSelector*
Add a button to the toolbar. Its bitmap can be found in the CPBitmapDict with the key *aSelector*. Its left button selector is *aSelector*. It will be *numSpaces* from its neighbor and its right button selector is *rbSelector*.

cellSize: *aPoint*
Set the size of the buttons. The default is 25@22.

currentIndex
Return the number of the button currently touched by the mouse. The buttons are number from left to right starting at one.

disableElements
Disable all of the buttons.

disableItem: *aSelector*
Disable the button with the left button selector *aSelector*.

enableElements
Enable all of the buttons.

enableItem: *aSelector*
Enable the button with the left button selector *aSelector*.

postAutomatic
Set the postAutomatic style. This will cause the buttons to remain pressed until the action they initiate finishes.

preAutomatic
Set the postAutomatic style. This will cause the buttons to return to their unpressed states before the action they initiate takes place.

rbSelectorAt: *aKey*
Returns the right button selector of the button with the index *aKey*.

selectedItem
Returns the index of the selected button.

selectItem: *aKey*
Select button with the index *aKey*.

selector
Returns the left button selector of the selected button.

selectorAt: *aKey*
Returns the left button selector of the button with the index *aKey*.

toggle
Set the toggle style. This will cause the buttons to remain pressed until another one is pressed.

Supported Events

select
This event occurs whenever a button is selected. Use the **selectedItem** or **selector** methods to query the index or selector of the selected button.

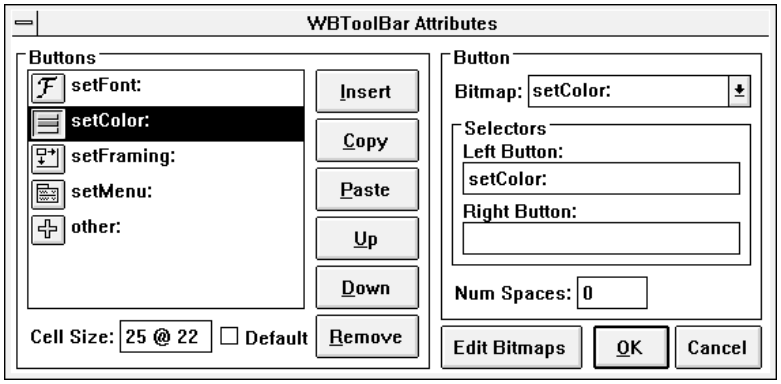
selecting
This event occurs whenever an enabled button is pressed while the mouse is down within the toolbar. A select event may also be generated if the mouse is released while in the button. Use the **currentIndex** method to query the current index.

doubleClick
This event occurs whenever a button is double clicked. Use the **selectedItem** or **selector** methods to query the index or selector of the selected button.

gettingFocus
This event occurs whenever the toolbar gets the focus.

showHelp
This event occurs whenever an button is touched while the mouse is down within the toolbar. This event occurs regardless of whether the button is enabled. Use the **currentIndex** method to query the current index.

WindowBuilder Extensions



The WBToolBar attribute editor provides everything you need to create and edit toolbars. The ListBox on the left side of the window shows the buttons that make up the toolbar and their corresponding left button selectors.

Below the list, the **Cell Size** field provides a place where the size of the buttons may be specified. For normal applications the default is 25@22. The **Default** CheckBox will set the cell size to the default and disable the cell size field.

The buttons to the right of the list are used to manipulate the toolbar buttons. The **Insert** button will insert a new button after the currently selected button. The **Copy** button will copy the current button. The **Paste** button will paste the previously copied button after the current button. The **Up** button will move the current button up while the **Down** button will move it down. Finally, the **Remove** button will delete the currently selected button.

On the right side of the editor can be found controls for manipulating the currently selected toolbar button. The **Bitmap** combobox can be used to select a toolbar button from the CPBitmapDict. *Only bitmaps designed to be used as toolbar buttons should be used.* Toolbar button bitmaps are defined with their up and down states side by side. They should be exactly twice as wide as the cell size used by the toolbar. If this is not the case, the toolbar buttons will not look very good. Fortunately, WindowBuilder Pro provides a mechanism for automatically generating button templates and editing them (See the section on the CPBitmapManager for further details).

Below the bitmap selection combobox are fields for specifying the **Left Button** and **Right Button** selectors to be used with the button. The left button selector will always initially default to the same name as the button. Ideally you would give your toolbar bitmaps names that correspond to the

functions they are meant to invoke. The right button selector is optional and is generally used to popup a menu for the button.

Below these selectors is the **Num Spaces** field used to specify how far (in pixels) the currently selected button should be from the one before it. The default is zero which will result in buttons that are right next to one another.

The **Edit Bitmaps** button at the bottom of the editor invokes the CPBitmapManager. From there you may create and edit any bitmaps. It provides special commands for specifically creating and editing toolbar bitmaps.

Chapter 11 — Windows and Dialogs

WindowBuilder Pro provides specific support for windows and dialogs in addition to subpanes. In this section of the manual, we will discuss several window-oriented events that you can use with WindowBuilder Pro. We will also describe the window and dialog attribute editors.

Supported Events

activate
This event occurs whenever a window becomes active. The active window has the input focus and displays its border using the active window border color. Note that a window becomes active when the user clicks on it *and* when it is first opened by the system.

close
This event occurs whenever a window receives the `close` message. Responding to this event gives you the opportunity to clean up before the window closes (i.e., closing dependents, freeing system resources, etc.). If there is no event handler for this event or if it returns nil, the closing process will continue normally. Returning anything else will interrupt the close process and the window will not close.

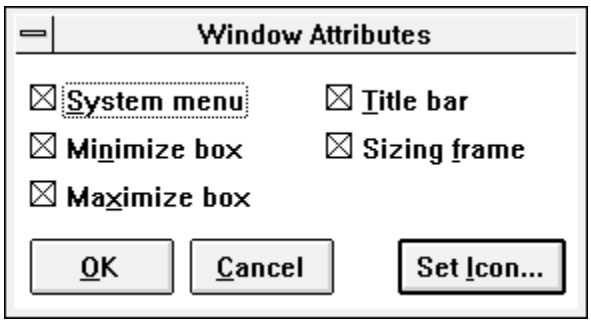
help
This event occurs whenever the F1 key is pressed. .You can handle this event yourself and display your own help windows or you can let the operating system handle it.

opened
This event occurs after the window and all of its subpanes have been built but before they are displayed to the user. This event can be used to perform any final initializations of the window or its subpanes.

timer
This event occurs whenever the window receives a time message from the operating system. For more information on setting up timers refer to Digitalk's documentation.

validated
This event occurs after the window has become a true operating system window but before its subpanes have been built or it has been displayed to the user.

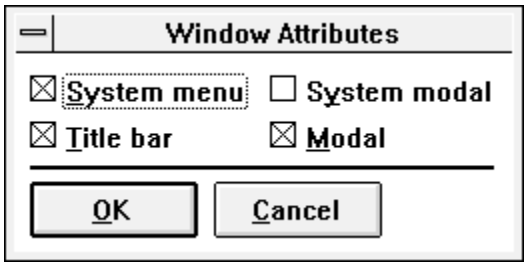
WindowBuilder Extensions



The Window attribute editor allows you to specify the frame characteristics of the window. A window may optionally have a system menu, a minimize box, a maximize box, a title bar, or a sizing frame.

Clicking the **Set Icon...** button will open a file dialog from which you may choose an icon file (.ICO) to attach to the window. This is the icon that will be used when the window is minimized.

☛ WindowBuilder Pro provides an easy way to attach icons to windows. There are other mechanisms to do this if you want to access icons from resource files. See Digitalk's documentation for further details.



The Dialog attribute editor allows you to specify the frame characteristics of a dialog. A dialog may optionally have a system menu or a title bar. The dialog box may also be modal (to the active window) or system modal (modal to all of the application's windows).

Chapter 12 — CompositePanes

This section will discuss how to create and write code for CompositePanes. CompositePanes can be virtually any combination of subpanes or other composite panes.

Creating CompositePanes

New compositepanes may be created in one of two ways. The **File** menu, **New CompositePane** command will create an empty compositepane to which you may add widgets. Alternatively, you may select a collection of widgets in the edit window and then use the **File** menu, **Create CompositePane** command (this command is also available from the popup group menu).

Editing the contents of a compositepane is handled like editing a window or dialog. Widgets may be added and event handlers may be specified for them.

When you are done specifying the compositepane it may be saved with the normal **File** menu, **Save** command. All compositepanes are subclasses of CompositePane or one of its subclasses. This is directly analogous to window and dialogs. Rather than a `createViews` method, compositepanes define an `addSubpanes` method. The `addSubpanes` method defines the layout and contents of the compositepane.

Event handling method stubs will also be created. You should flesh them out in the same way you would complete a window or dialog event handling stub.

Styles

CompositePanes may be displayed in several styles. The default style causes them to display without a border or any scrollbars. The `borders` style results in a box around the compositepane. The `scrollbars` style gives the compositepane vertical and horizontal scrollbars. The `verticalScrollBar` style adds a single vertical scrollbar.

Nesting

CompositePanes may be nested arbitrarily deep. Very powerful interface objects can be created by building up layers of compositepanes.

Warning: be careful not to define a compositepane to include a copy of itself. WindowBuilder Pro will stop you from adding a compositepane to itself, but it cannot check for deeper levels of recursion (e.g., A contains B which contains A).

Tab Order

The tab order that should be in effect within the compositepane may also be specified. When compositepane are nested or when they are used in a window this tab order is properly maintained (compositepanes acts like single objects in the Tab editor of their containing windows).

Adding Events

When creating a compositepane it is often useful to define events that are specific to the compositepane that can be exported to its owner (generally the window in which it resides). This also provides an opportunity to rename the real events to something more meaningful.

Adding events to a compositepane is simple. First define a class method for the compositepane called `supportedEvents`. This method should call `super supportedEvents` and then add its own events (defined as Symbols). An example from the **OkCancelPane** example provided with WindowBuilder Pro would look like:

```
supportedEvents
   ^super supportedEvents
      add: #ok;
      add: #cancel;
      yourself.
```

This example exports two events, `ok` and `cancel`. Actually generating these events is the second step. Generating an event is easy. The code is of the form `self event: #eventname`. Looking at the OkCancelPane example again, we see that it converts `clicked` events on its two buttons to `ok` and `cancel` events for its owner to use.

For example, if the **OK** button in the OkCancelPane performs the message `ok` when it receives a `clicked` event, the code for the `ok` method would look like:

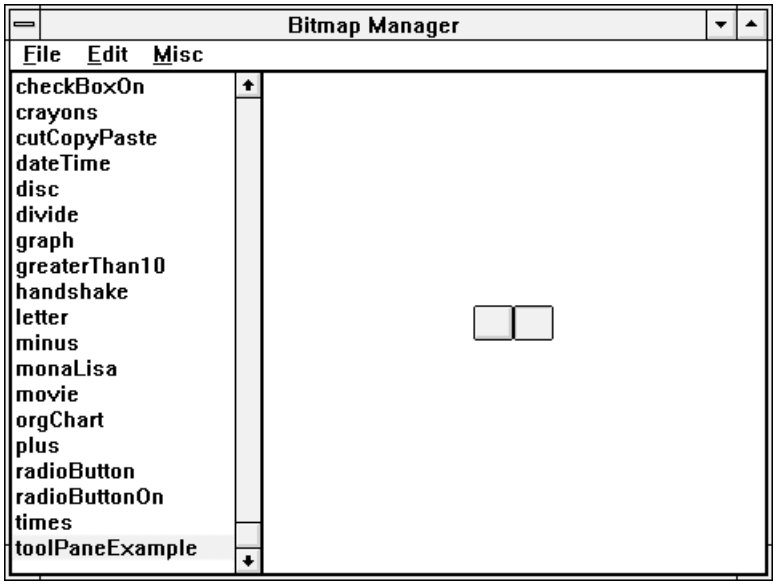
```
ok: aPane
   self event: #ok
```

The owner window could then respond to the `ok` event of the OkCancelPane directly rather than a `clicked` event on the button itself.

For further details on adding your own events to CompositePanes, it is beneficial to examine some of the other CompositePane examples provided with WindowBuilder Pro.

Chapter 13 — Bitmap Manager and Button Editor

The CPBitmapManager is an application included with WindowBuilder Pro which provides a convenient way to create, edit, store, and retrieve bitmaps within the Smalltalk environment.

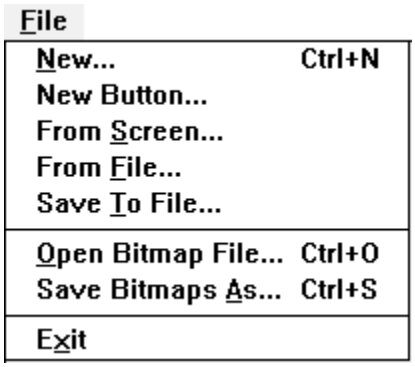


The bitmap manager application consists of a list of the managed bitmaps, a display pane for the currently selected bitmap, and a command menubar for creating, editing, storing, and retrieving bitmaps.

Each bitmap stored in the bitmap manager has a name associated with it. The list on the left contains the names of all the bitmaps being handled by the bitmap manager. When a name is selected, the bitmap with that name will appear on the right.

Most of the operations you will do with the bitmap manager will be executed from the menubar at the top of the window. The commands available are described below

File Menu



New…
This command creates a new bitmap. When you create a new bitmap, a dialog will appear asking you name of the bitmap, its size, and whether the bitmap should be color or black and white. After dismissing it, a blank bitmap will be added to the listbox below.

New Button…
This command creates a new button bitmap for use with a WBToolBar. When you create a new button bitmap, a dialog will appear asking you the name of the button and its size. After dismissing it, a blank button bitmap will be added to the listbox below. Use the **Edit | Button Edit...** command to easily edit the button.

From Screen…
This command allows you to create a bitmap by interactively capturing a rectangular area from the screen. When this command is executed, the cursor will change to indicate that you should click and drag across a rectangular area. When you release the mouse, you will be prompted for a name for this bitmap, and the section of the screen you selected will appear in the bitmap manager.

As a convenience for creating toolbar buttons this command may be used to load the left and right sides of a button template. With the button you want to fill selected, hold the **ALT** key down while selecting this command. Any image that you capture will be scaled and inserted into the current button. *Do not* use this feature with a non-button bitmap.

From File…
This command allows you to import a bitmap from a .bmp or .ico file. It

will prompt you for a file to read in, and a name for the bitmap contained in it. After reading it in, the bitmap will appear in the bitmap manager.

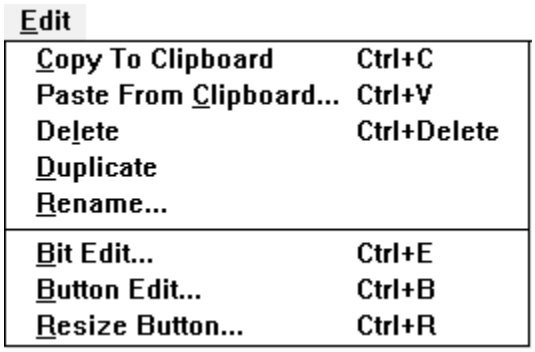
Save To File…
This command allows you to save the selected bitmap to disk as a .BMP file.

Open Bitmap File…
This command allows you to read in a list of bitmaps from a file that have been saved using the **Save Bitmaps As…** command (described below). It is very useful for cross-image bitmap management. When executed, this command will bring up a file dialog, prompting you to select a bitmap file. BitmapManager bitmap files must have the extension '.BDT'. If you hold the **ALT** key down while executing this command, the bitmaps from the .BDT file will be merged with the current bitmaps (rather than replacing them).

Save Bitmaps As…
This command allows you to export the entire list of bitmaps in the bitmap manager to a file. This is very useful for cross-image bitmap management. The bitmaps and their names will be saved in a file with the extension '.BDT', in a format suitable for the **Open Bitmap File…** command (described above). If you hold the **ALT** key down while executing this command, you will be able to select a subset of the bitmaps in the dictionary to export to the file. *In order to create a runtime image you will need to export your bitmaps from your development image and then import them into the runtime image using the commands presented at the end of this section.*

Exit
Closes down the bitmap manager. This can also be accomplished by double-clicking on the system menu of the window.

Edit Menu



Copy To Clipboard
This command allows you to export bitmaps to the Windows clipboard. When executed, it will place a copy of the currently selected bitmap in the clipboard.

Paste From Clipboard…
This command allows you to import bitmaps from the Windows clipboard. When this command is executed, the bitmap manager will prompt you for a bitmap name. After giving it a name, the bitmap in the clipboard will appear in the bitmap manager.

Delete
This command allows you to remove a bitmap from the bitmap manager. When executed, it will verify that you want to remove the selected bitmap, then remove it.

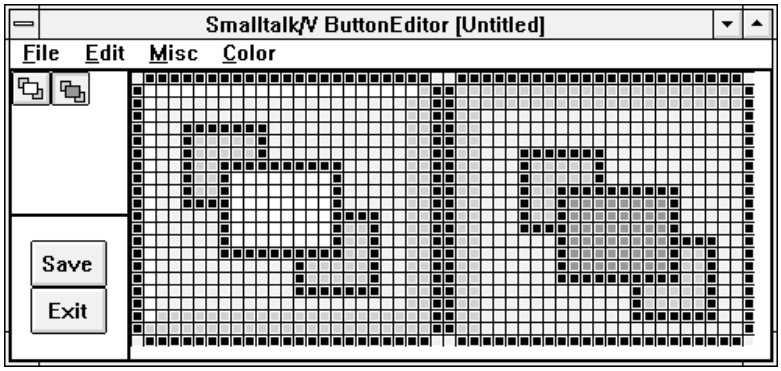
Duplicate
This command allows you to create new bitmaps based on existing ones. It will prompt you for a new name, then create a copy of the currently selected bitmap with that name.

Rename…
This command allows you to change the name of a bitmap within the bitmap manager. When executed, it will prompt you for a new name for the currently selected bitmap.

Bit Edit…
This command allows you to make use of Digitalk's bit editor to change the appearance of a bitmap. When executed, this command will launch Digitalk's bit editor on the currently selected bitmap. After making

changes to the bitmap within the bit editor, press the **Save** button, and the changes will be propagated to the bitmap manager.

Button Edit…
This command allows you to make use of a modified version of Digitalk's bit editor to change the appearance of a button bitmap. When executed, this command will launch the button editor on the currently selected button bitmap. After making changes to the bitmap within the bit editor, press the **Save** button, and the changes will be propagated to the bitmap manager.



The Button Editor works exactly like Digitalk's Bit Editor with one exception. As you place pixels in the left half of the button (the **Up** side), these pixels will be offset and mirrored on the right side (the **Down** side). Color fills do not cause this to happen. After editing the left side, you can make subtle alterations to the right side to indicate the depressed state.

Resize Button...
This command allows you to intelligently resize the currently selected button. After you enter the desired size, a new button template will be created and the contents of the old button will copied over. If the new button is smaller than the old button, the contents will be scaled. If the new button is larger, the contents will be centered in the new button (holding down the **CTRL** key will cause the contents to be scaled up). Holding down the **ALT** key while executing this function will resize *all* buttons of the same size as the current button. This command is very useful when migrating applications from one screen resolution to another.

Programmatic Access

Bitmaps created with the bitmap manager can later be accessed by name within your code.

Access is provided through class methods in the class CPBitmapDict. For example, to retrieve a bitmap named ‘test’ that you’ve created with the CPBitmapManager, you would access it as follows:

```
theBitmap := CPBitmapDict at: 'test'.
```

To load the bitmap dictionary programatically do the following:

```
CPBitmapDict loadBitmaps: 'BITMAPS.BDT'
```