# Advanced VisualAge Development

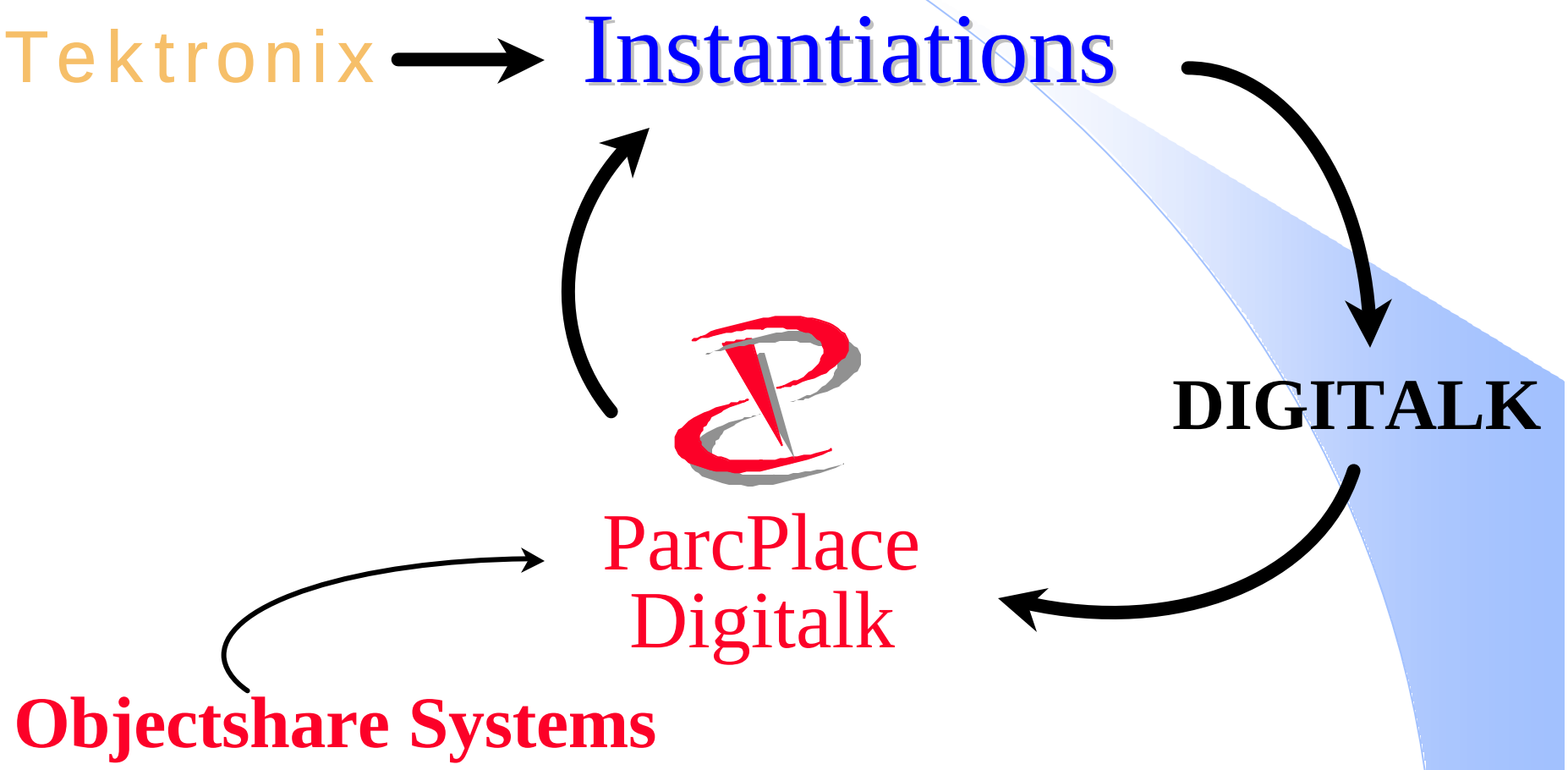
Eric Clayberg
Sr. Vice President of Product Development
Instantiations, Inc.
April 9, 2001

clayberg@instantiations.com
http://www.instantiations.com
978-750-3621

# Who Am I?

- First used Smalltalk in late ‘80s; full-time since 1991
- Co-Founder of (the original) ObjectShare in 1992
- Developer & Chief Architect of WindowBuilder Pro and over a dozen other commercial Smalltalk add-on products (VA Assist Pro, WidgetKits, etc.)
- Vice President of Development for ParcPlace-Digitalk 1996-97
- Sr. Vice President of Product Development for Instantiations 1997-present
- Former Smalltalk Editor for VA Magazine
- Usenet Junkie

# Who Is Instantiations?

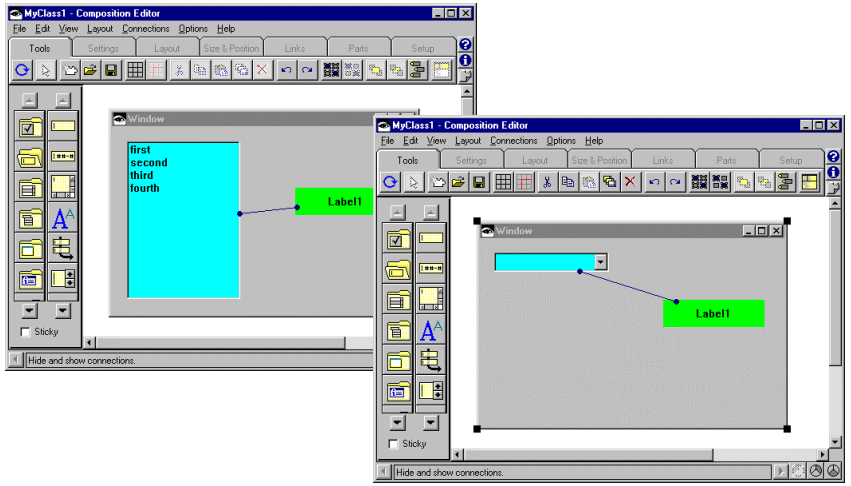


# Tutorial Roadmap

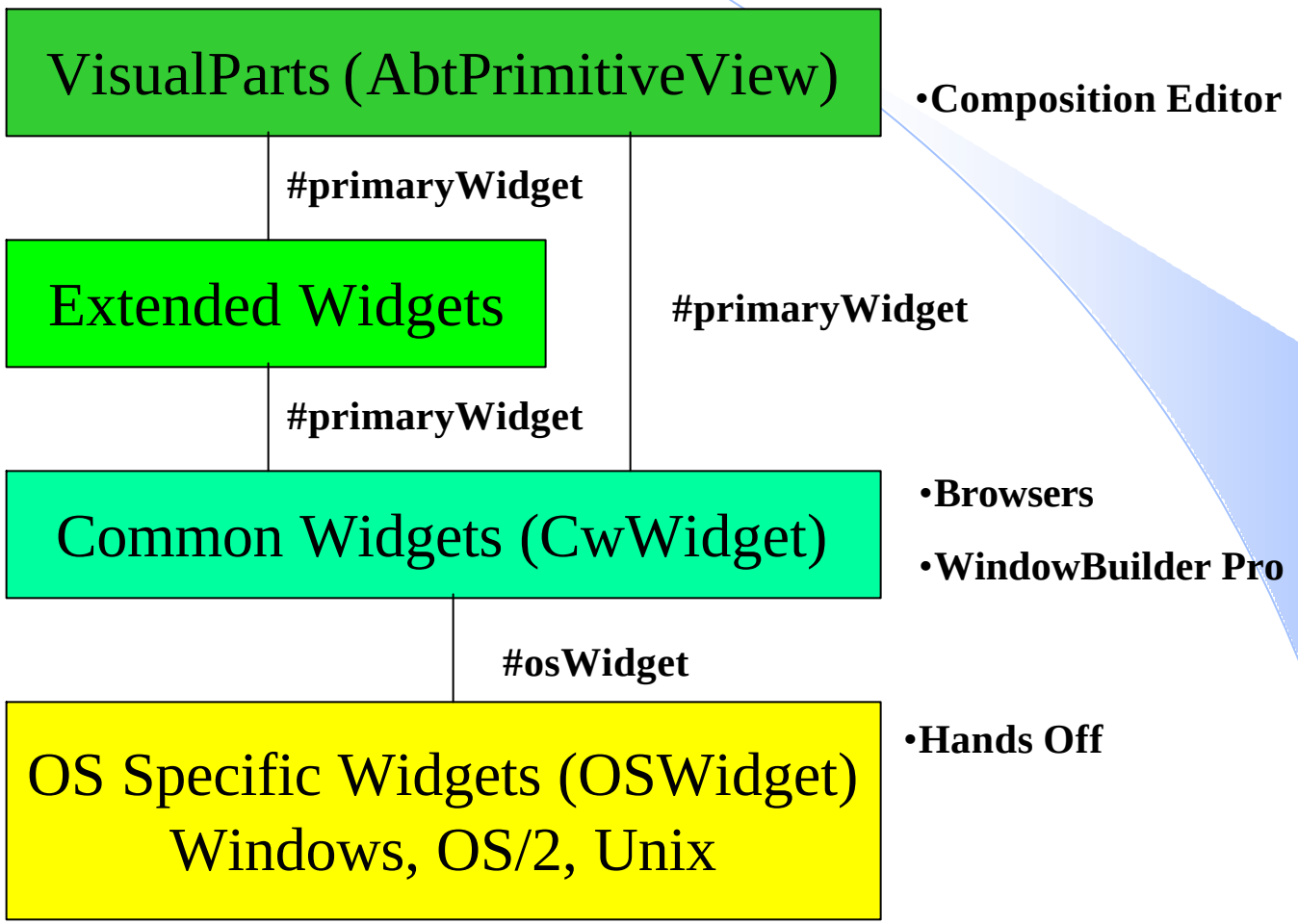
- Advanced GUI Techniques
- Custom Widget Development
- Complex Configuration Management
- Development Tool (Browser) Enhancements

# Advanced GUI Techniques

- Widget Callbacks & Event Handlers
- Using Pointer Motion
- Using Timers & Delays
- Determining Key State
- Creating Screen Snapshots
- Printing Images
- Clipboard
- Attachments
- Morphing

# Structure of VisualAge Widgets



# Widget Callbacks & Event Handlers

- Why use Callbacks and Events?
  - Abt layer exposes a subset of available protocol
  - More control
  - Create complex interactions
- What is the different between callback and event handlers?
  - Not much
  - Syntactically similar
  - Events are low-level occurrences like mouse down/up/move, pointer motion, key press/release, etc. Events are generated constantly (and may not be meaningful)
  - Callbacks are higher-level occurrences that imply some semantic meaning like button clicks, text input, focus changes
  - Abt-layer “events” are very high-level occurrences that wrapper a subset of Cw-layer callbacks
- Given an AbtPart, how do you get to its CommonWidget component?
  - Send the #primaryWidget message to the part
  - Do this in a method overriding #openInShellView or in a script triggered by the #openedWidget event (prior to opening, the primary widget is nil)

# Setting up a Callback Handler

- Use the #addCallback:receiver:selector:clientData: method
  - First parameter - the name of the callback (e.g., XmNactivateCallback)
  - “receiver” - the object to send the callback message to
  - “selector” - the 3-parameter message selector to send
  - “clientData” - an object to be passed to the receiver of the callback message as the clientData parameter when the callback is invoked, or nil
  - Example:

```
<cwWidget> addCallback: XmNactivateCallback
           receiver: self
           selector: #clicked:clientData:callData:
           clientData: nil
```

- Create the handler method
  - First argument - the widget that triggered the event
  - “clientData” - the object specified when the callback was set up (usually nil)
  - “callData” - data specific to the specified callback type
  - Example:

```
clicked: aWidget clientData: clientData callData: callData
           System message: ‘Hello World’
```

# Setting up an Event Handler

- Use the #addEventHandler:receiver:selector:clientData: method
  - First parameter - an integer event mask identifying the desired events. One or more of the following OR'ed together:
    - KeyPressMask - Keyboard down events
    - KeyReleaseMask - Keyboard up events
    - ButtonPressMask - Pointer button down events
    - ButtonReleaseMask - Pointer button up events
    - PointerMotionMask - All pointer motion events
    - Button1MotionMask - Pointer motion while button 1 down
    - Button2MotionMask - Pointer motion while button 2 down
    - Button3MotionMask - Pointer motion while button 3 down
    - ButtonMotionMask - Pointer motion while any button down
    - ButtonMenuMask - Menu request events
  - "receiver" - the object to send the event handler message to
  - "selector" - the 3-parameter message selector to send
  - "clientData" - an object to be passed to the receiver of the event handler message as the clientData parameter when the event handler is invoked, or nil
  - Example:

```
<cwWidget> addEventHandler: KeyPressMask | KeyReleaseMask
           receiver: self
           selector: #keyPressed:clientData:callData:
           clientData: nil
```

# Callback/Event Handler Tricks

- Use 3-argument blocks to avoid the need for handler methods
  - Block arguments should be “widget”, “clientData” and “callData” (or any name if you don’t care)
  - The “selector” should be #value:value:value:
  - Example:

```
<cwWidget> addCallback: XmNactivateCallback
           receiver: [:widget :clientData :callData |
                    System message: ‘Hello World’]
           selector: #value:value:value:
           clientData: nil
```

# Callback/Event Handler Tricks - 2

- Support unary & 1-argument callback handlers (like VSE)
  - Add the following method to CwPrimitive (and CwComposite) to override the CwBasicWidget>>addCallback:receiver:clientData: method

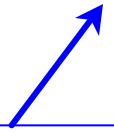
```
addCallback: callbackName receiver: receiver selector: selector
 clientData: clientData
    selector argumentCount <= 1
       ifTrue: [
         super
           addCallback: callbackName
           receiver: (selector argumentCount == 0
             ifTrue: [:a :b :c |
                receiver perform: selector]
             ifFalse: [:a :b :c |
                receiver perform: selector with: clientData value])
           selector: #value:value:value:
           clientData: clientData]
      ifFalse: [
        super
           addCallback: callbackName
           receiver: receiver
           selector: selector
           clientData: clientData]
```

# Callback/Event Handler Tricks - 3

- Now you can set up callback handlers like this:

```
buttonWidget addCallback: XmNactivateCallback
             receiver: self
             selector: #clicked
             clientData: nil

listWidget addCallback: XmNsingleSelectionCallback
             receiver: self
             selector: #selected:
             clientData: [listWidget selectedItem]
```



the argument to the #selected: method

# Using Event Handlers

- What are some useful things you can do with event handlers?
  - Detect clicks on static labels (using ButtonReleaseMask )
  - Detect when the mouse passes over a widget (using PointerMotionMask)
    - Implement hover/balloon help
    - Implement simple status line help
- Example (click on a static label)
  - Add the following event handler to a CwLabel

```
<aCwLabel> addEventHandler: ButtonReleaseMask
           receiver: self
           selector: #clicked:clientData:callData:
           clientData: nil
```

  - Implement the #clicked:clientData:callData: method

```
clicked: widget clientData: clientData callData: callData
           System message: 'I''m clicked'
```

# Using Pointer Motion

- Example (status line help)
  - Add the following event handler to every widget in the window (including the main form so that you can detect when the pointer isn't over a widget)

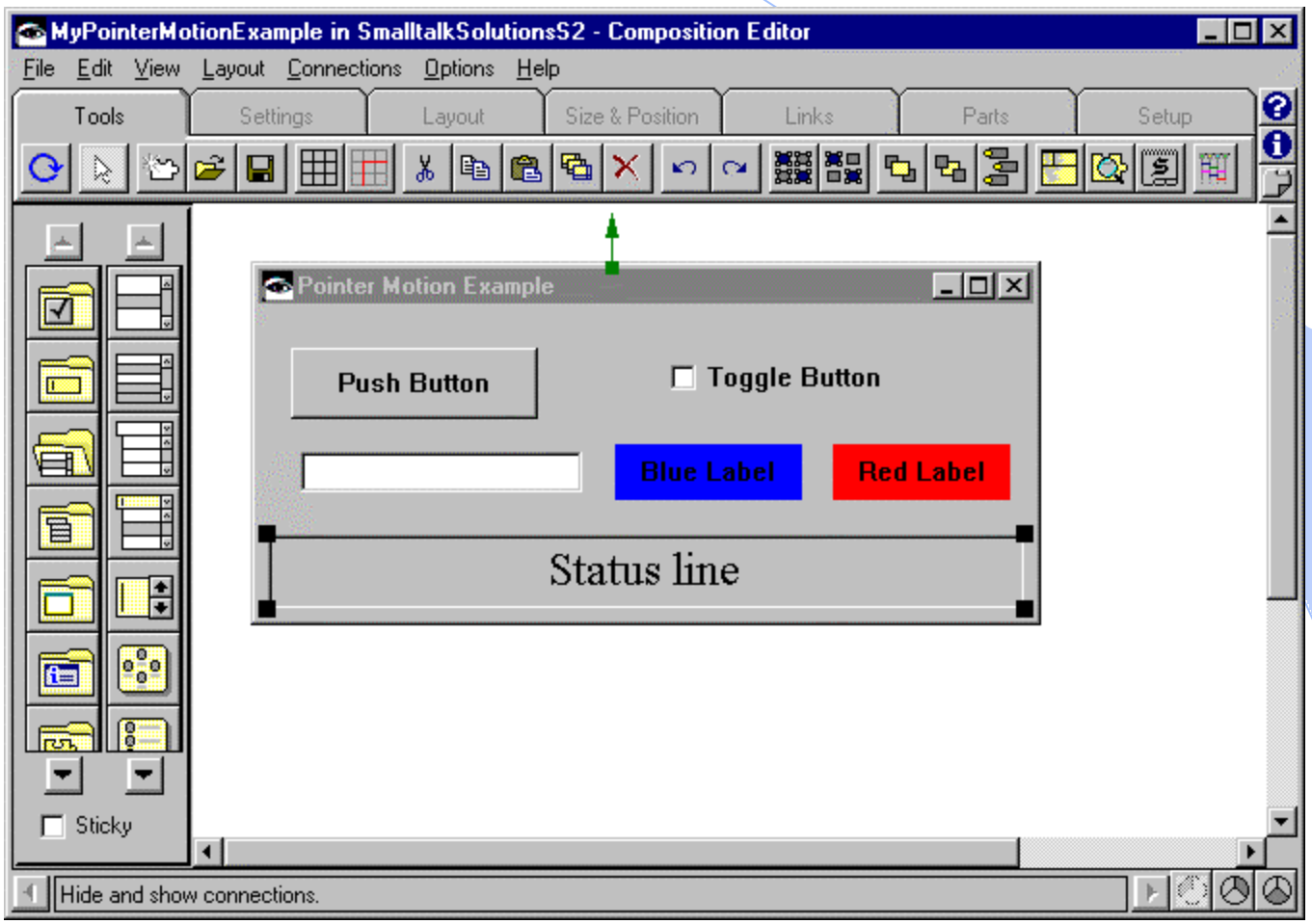
```
<aCwWidget> addEventHandler: PointerMotionMask
            receiver: self
            selector: #pointerMotion:clientData:callData:
            clientData: nil
```

  - Add a static text label name "statusLine" to the bottom of the window
  - Implement a dictionary named "helpDict" that maps widget names to help text
  - Implement the #pointerMotion:clientData:callData: method

```
pointerMotion: widget clientData: clientData callData: callData
            self statusLine labelString:
                    (self helpDict at: widget name)
```

  - Let's build it...

# Pointer Motion Example



# Using Delays

- Goal: execute some code after a fixed amount of time
- Solutions
  - Use a Delay
  - Use a Timer
- Example:
  - Delay for one second and then execute some code

```
(Delay forMilliseconds: 1000) wait.
self doSomething
```

  - Problem: blocks the current process
  - Solution: fork the code as a background process:

```
[(Delay forMilliseconds: 1000) wait.
self doSomething] forkAt: Processor userBackgroundPriority
```

- Example: Ring Bell every second for five seconds

```
5 timesRepeat: [
     (Delay forMilliseconds: 1000) wait.
     CgDisplay default bell: 0].
```

# Using Timers

- Create a one shot timer
  - Use the CwAppContext>>addTimeout:receiver:selector:clientData:
  - First argument - integer specifying the time interval in milliseconds
  - “receiver” - the object which is the receiver of the work procedure message
  - “selector” - the Symbol which is the 1-parameter message selector to send.
  - “clientData” - any object which is to be passed as the parameter to the work procedure message
  - Example:

```
CwAppContext default
        addTimeout: 1000 “one second”
        receiver: [:clientData | CgDisplay default bell: 0]
        selector: #value:
        clientData: nil
```

# Using Timers - 2

- Create a recurring timer to update a clock
  - Create a static text widget named “clock”
  - Create a #startClock: method

```
startClock: milliseconds
    self clock labelString: Time now printString.
    CwAppContext default addTimeout: milliseconds
        receiver: self
        selector: #updateClock:
        clientData: milliseconds
```

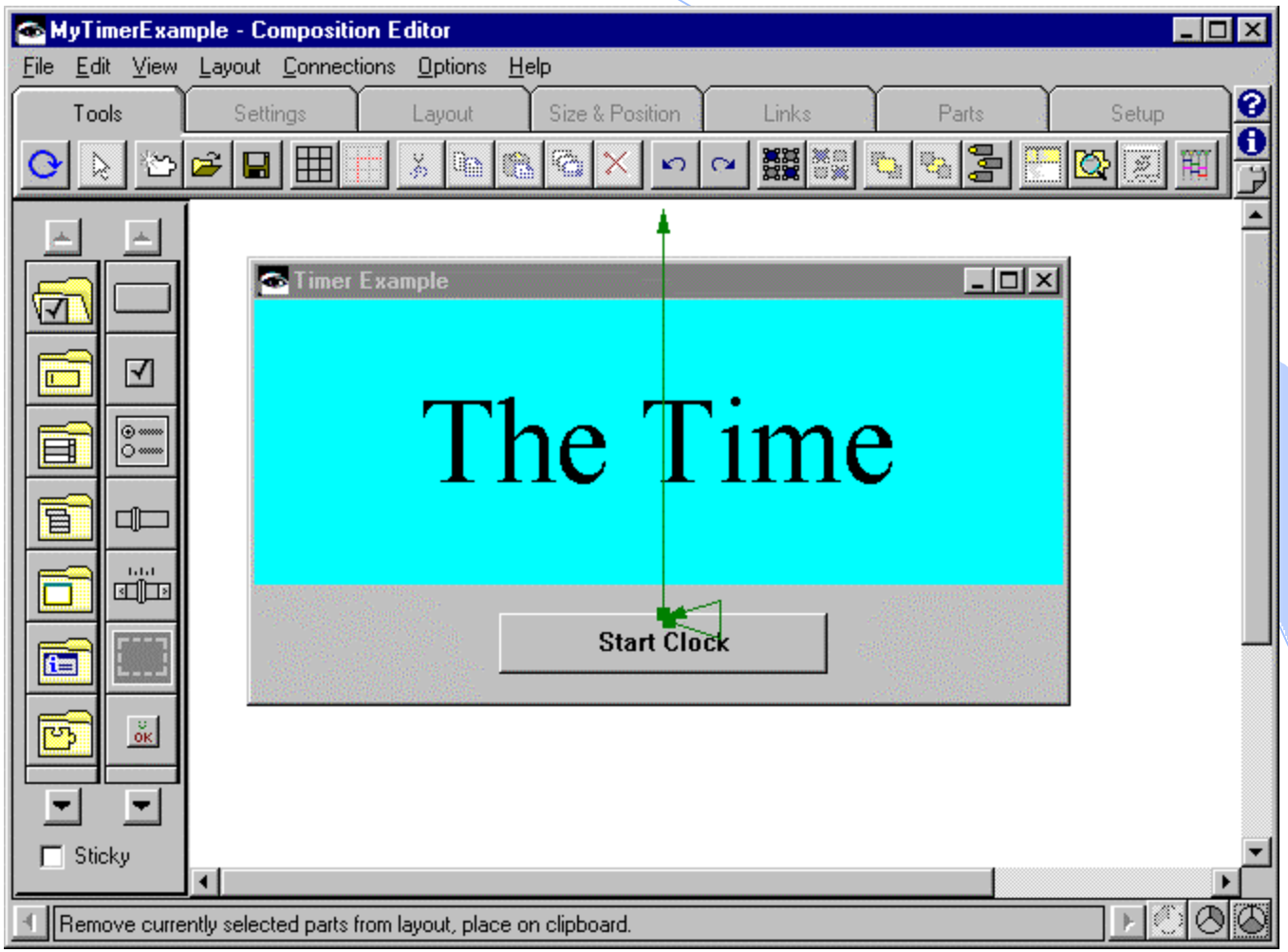
  - Create an #updateClock: method

```
updateClock: milliseconds
    self clock isMapped ifFalse: [^self].
    self clock labelString: Time now printString.
    CwAppContext default
        addTimeout: (milliseconds -
            (Time millisecondClockValue \\ milliseconds))
        receiver: self
        selector: #updateClock:
        clientData: milliseconds
```

  - Start the clock so that it updates every second:

```
self startClock: 1000
```

# Clock Example



# Another Way to Delay

- Use the CwAppContext>>asyncExecInUI: (aBlock) method
  - A favorite “magic” method for executing a block code after a short delay
  - Technically, what does it do?
    - Evaluates aBlock in the UI Process. No result is returned.
    - Processes with higher priority than the UI will NOT block.
    - In this case, aBlock is executed the next time the UI becomes active.
    - If this message is sent by the UI process, then aBlock will be executed after all previously queued background graphic requests have been executed
    - Example:

```
...
CwAppContext default asyncExecInUI:
      [Transcript cr; show: ‘1’].
Transcript cr; show: ‘2’.
...
```

  - Result:

```
...
2
1
...
```

# Determining Key State

- Why would you need to do this?
  - Constrain behavior (e.g., Extended Select List Boxes)
  - ALT-key hacks
  - Conditional breakpoints
- How do you determine whether an arbitrary modifier key is depressed?
  - Look at the CgDisplay>>osGetModifierState method
    - Can be sent to any CgDisplay instance at any time. For example:
      ```
      CgDisplay default osGetModifierState
      ```
    - Returns an integer encoding the key state
    - Use the Integer>>anyMask: method to test for different keys (returns a Boolean)
  - Examine the event hander data

# Determining Key State - 2

- Useful methods to add to Object
  - Is any key down?
    **isKeyDown: keyMask**
    ```
        CgDisplay default osGetModifierState anyMask: keyMask
    ```
  - Is Alt key down?
    **isAltKeyDown**
    ```
        self isKeyDown: CwConstants::Mod1Mask
    ```
  - Is Ctrl key down?
    **isControlKeyDown**
    ```
        self isKeyDown: CwConstants::ControlMask
    ```
  - Is Shift key down?
    **isShiftKeyDown**
    ```
        self isKeyDown: CwConstants::ShiftMask
    ```
  - Is Caps Lock key down?
    **isCapsLockKeyDown**
    ```
        self isKeyDown: CwConstants::LockMask
    ```
  - Is Left Mouse Button down?
    **isLeftMouseButtonDown**
    ```
        self isKeyDown: CwConstants::Button1Mask
    ```

# Creating Screen Snapshots

- Why is this useful?
  - Useful for creating documentation
  - Runtime error reporting
  - Simple reports
- Here's a handy method for creating a pixmap from any window
  - The OSWidget>>screenRect method answers the rectangle of the receiver in screen coordinates (this is different from the CwWidget boundingBox method which answers the inner bound)
  - The CgDrawable>> createPixmap:height:depth: method create a pixmap (bitmap)
  - The CgDrawable>>copyArea:gc:srcX: srcY:width:height:destX:destY: method copies an area of one image into another

```
CwShell>>getSnapshot
    | rect defWin pixmap |
    rect := self osWidget screenRect.
    defWin := CgWindow default.
    pixmap := defWin
      createPixmap: (rect right - rect left) abs
      height: (rect bottom - rect top) abs
      depth: defWin depth.
    defWin
      copyArea: pixmap
      gc: CgGC default
      srcX: rect left
      srcY: rect top
      width: (rect right - rect left) abs
      height: (rect bottom - rect top) abs
      destX: 0
      destY: 0.
    ^pixmap
```

# Copying Graphics to the Clipboard

- Once we have the screen snapshot, it would be nice to do something with it
- Here's a handy method for copying a pixmap to the clipboard
  - The CgDisplay>>clipboardStartCopy: clipLabel:itemIdReturn: method message sets up storage and data structures to receive clipboard data
  - The CgDisplay>>clipboardCopy: itemId:formatName:buffer:privateId: method copies a data item to temporary storage
  - The CgDisplay>>clipboardEndCopy: itemId: method locks the clipboard from access by other applications, places data in the clipboard data structure, and unlocks the clipboard

```
CgPixmap>>copyToClipboard
    | defaultDisplay window itemId |
    defaultDisplay := CgDisplay default.
    window := CgWindow default.
    itemId := ReturnParameter new.
    defaultDisplay
          clipboardStartCopy: window
          clipLabel: 'Pixmap Copy'
          itemIdReturn: itemId.
    defaultDisplay
          clipboardCopy: window
          itemId: itemId value
          formatName: 'PIXMAP'
          buffer: self
          privateId: 0.
    defaultDisplay
          clipboardEndCopy: window
          itemId: itemId value.
```

# Copying Text to the Clipboard

- The same technique works for text as well
- Here's a handy method for copying a text to the clipboard

```
EsString>>copyToClipboard
    | display window itemId |
    display := CgDisplay default.
    window := CgWindow default.
    itemId := ReturnParameter new.
    display
          clipboardStartCopy: window
          clipLabel: 'Text Copy'
          itemIdReturn: itemId.
    display
          clipboardCopy: window
          itemId: itemId value
          formatName: 'STRING'
          buffer: self
          privateId: 0.
    display
          clipboardEndCopy: window
          itemId: itemId value.
```

# Printing Images

- Just in case you want to know how to print a Pixmap, here's how to do it:

```
CgPixmap>>copyToPrinter
    | image printDisplay printerShell default prompter |
    CgDisplay allPrinterDisplayNames isEmpty
        ifTrue: [^System message: 'There are no printers available.'].
    (prompter := CwPrinterPrompter new) prompt isNil ifTrue: [^self].
    image := self getDeviceIndependentImage: self rectangle.
    default := prompter displayName.
    printDisplay := CwAppContext default
          openDisplay: default
          applicationName: 'Print Pixmap'
          applicationClass: nil.
    printerShell := CwPrinterShell
          appCreateShell: 'Printer Shell'
          applicationClass: nil
          display: printDisplay
          argBlock: [:w | w jobAttributes: prompter jobAttributes].
    ...
```

# Printing Images - 2

```
CgPixmap>>copyToPrinter continued:

    ...
    printerShell
        addCallback: XmNmapCallback
            receiver: [:shell :clientData :callData |
                printerShell startJob
                    ifTrue: [printerShell startPage]
                    ifFalse: [printerShell destroyWidget]]
            selector: #value:value:value:
            clientData: nil.
    ...
```

# Printing Images - 3

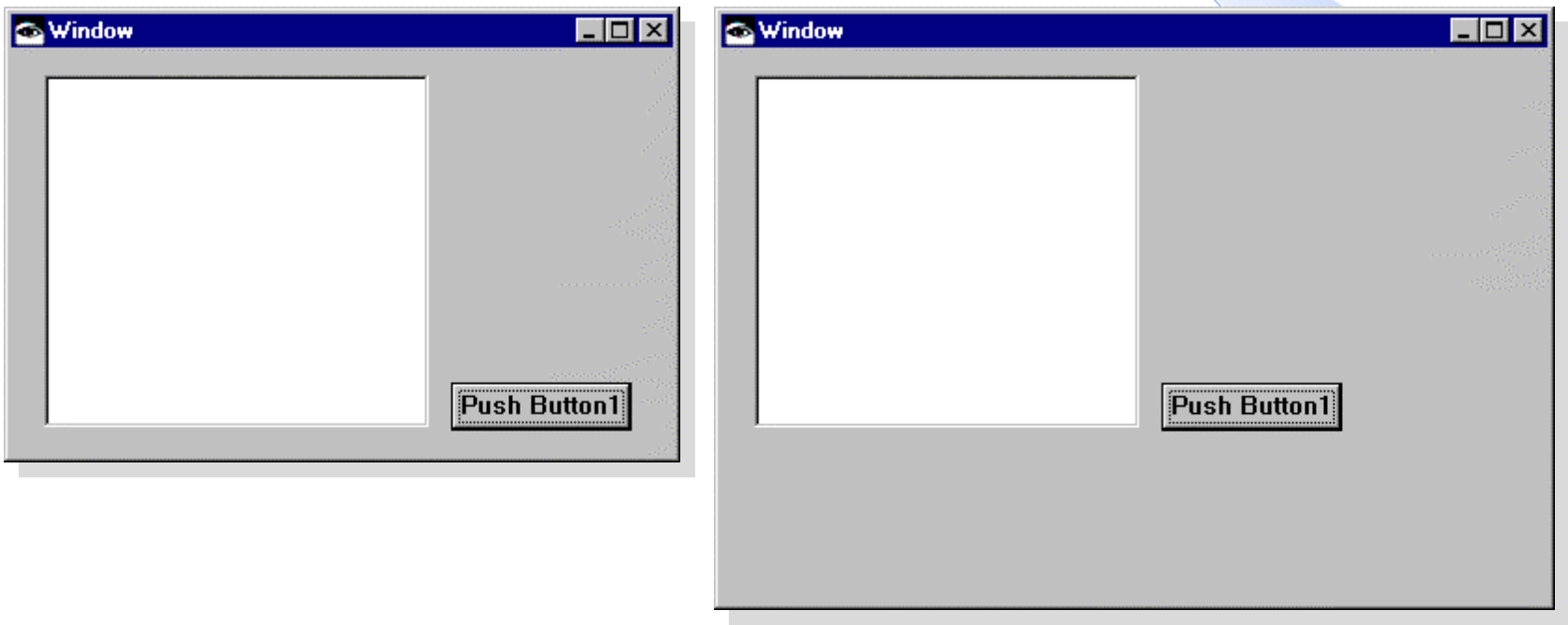
**CgPixmap>>copyToPrinter continued:**

```
...
printerShell
addCallback: XmNexposeCallback
    receiver: [:shell :clientData :callData | | scale printGC |
        scale := printerShell width / image width
            min: printerShell height / image height.
        printGC := printerShell window createGC: 0 values: nil.
        printerShell window putDeviceIndependentImage: printGC
            image: image
            srcRect: (0 @ 0 extent: image extent)
            destRect: (0 @ 0 extent: (image extent * scale) truncated).
        printGC freeGC.
        printerShell
            endPage;
            endJob;
            destroyWidget.
        printerShell display close]
    selector: #value:value:value:
    clientData: nil.
printerShell realizeWidget
```

- Thus you can print any screen like this:
  **`Transcript shell getSnapshot copyToPrinter`**

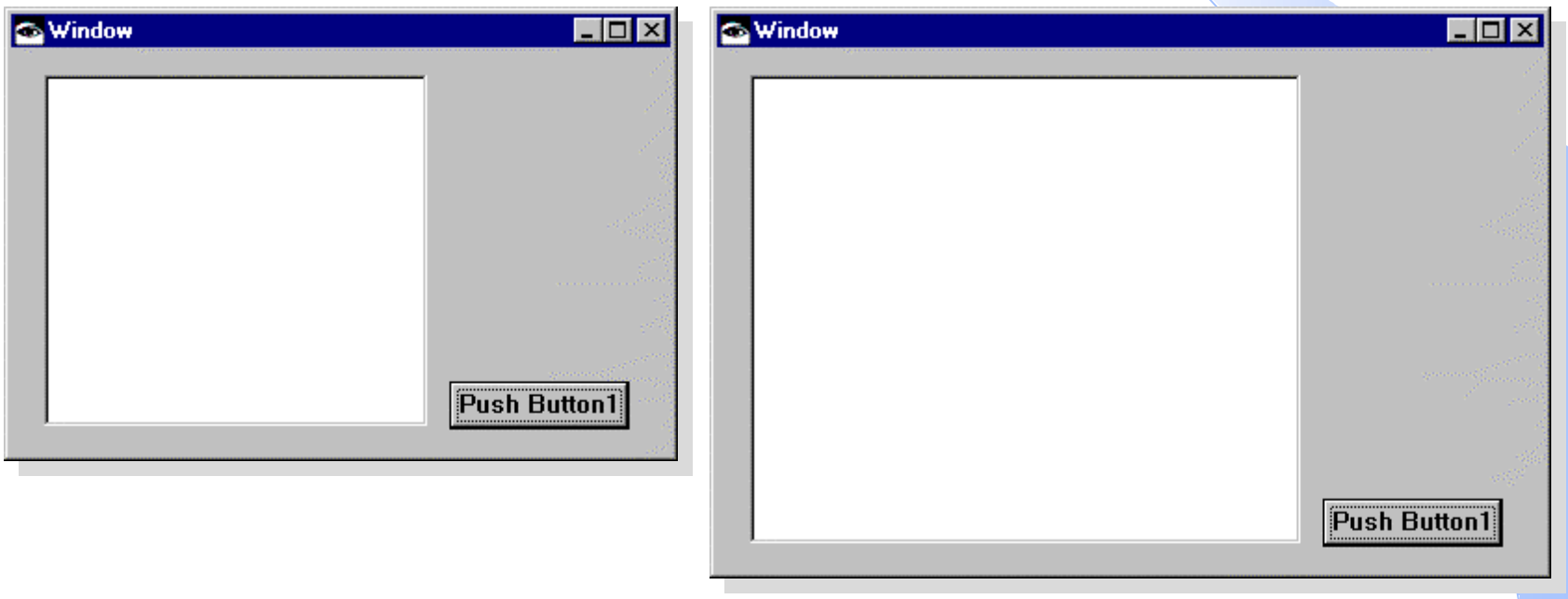
# Attachments

- By default all widgets are locked to the upper left corner of a window
- For example:



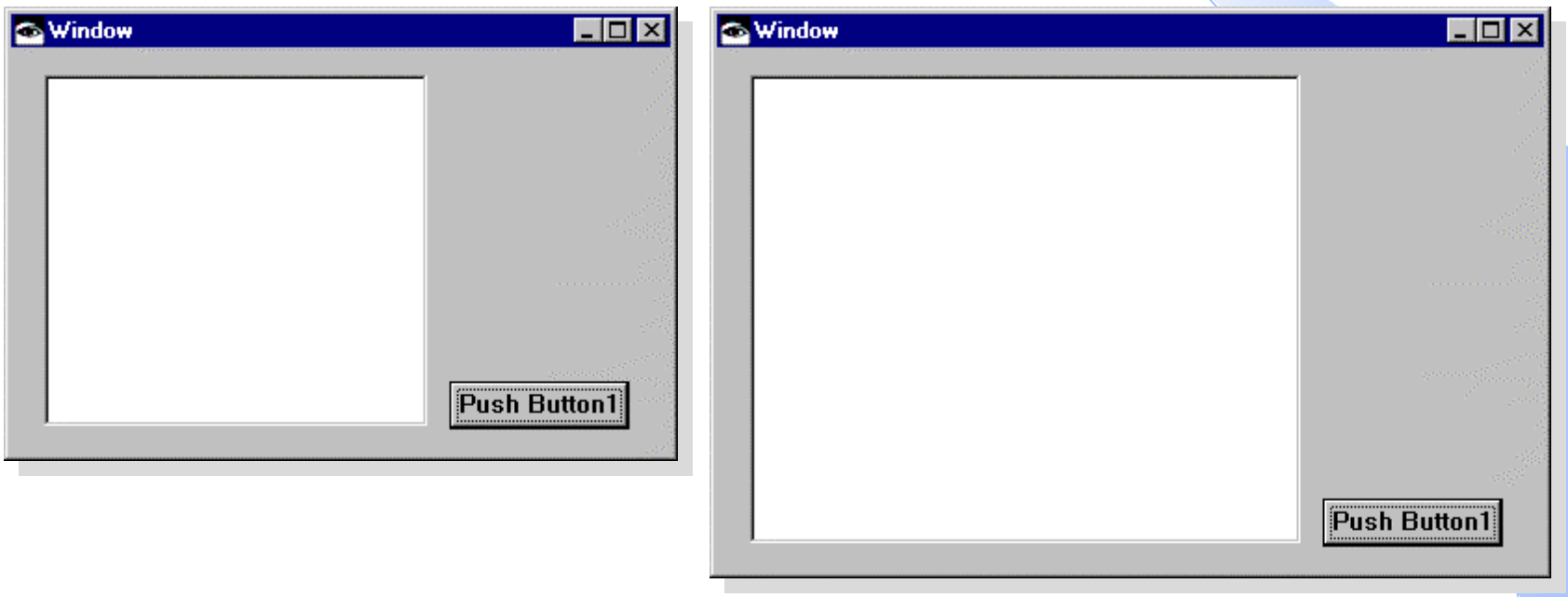
# Attachments - The Ideal

- Ideally, we would like to specify what happens to each widget when the window resizes



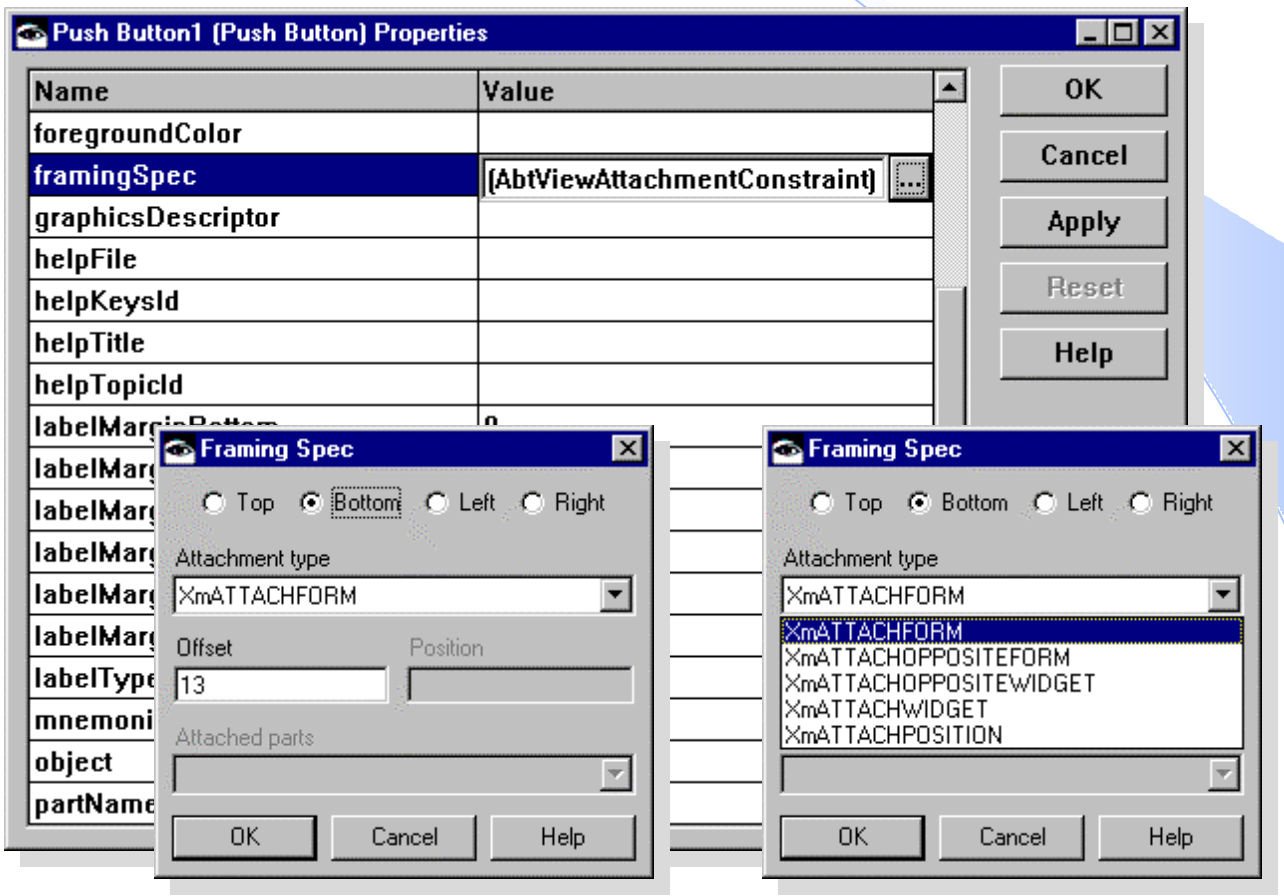
# Attachments - The Ideal

- For example:



# Attachments - VA Editor

- Here's the lame attachment editor supplied with VisualAge



# Attachments - Sample Code

- With very little effort, we can dramatically simply the process
  - There are hundreds of possible attachment combinations
  - But only a few (10-20) that are commonly used
  - By optimizing those cases, we can dramatically speed up the GUI layout process
- Sample code to add a “Set Attachments” cascaded menu to the popup widget menu in the Composition Editor
  - Add the following method to AbtPrimitiveView (and AbtCompositeView)

```
abtAddOwnItemsToPopUpMenu: aPopUpMenu for: anEditPart
    super abtAddOwnItemsToPopUpMenu: aPopUpMenu for: anEditPart.
    anEditPart addAttachmentItemsToPopUpMenu: aPopUpMenu
```

# Attachments - Sample Code 2

- Add the following methods to AbtCwEditPart

```
attachAllSides
  self performBlockedUpdate: [| fs |
    (fs := self visualPolicy visualPartFramingSpecTranslateBy: 0@0)
      leftEdge: (fs leftEdge attachment: XmATTACHFORM currentView: self part);
      rightEdge: (fs rightEdge attachment: XmATTACHFORM currentView: self part);
      topEdge: (fs topEdge attachment: XmATTACHFORM currentView: self part);
      bottomEdge: (fs bottomEdge attachment: XmATTACHFORM currentView: self part).
    self frameVisualPart: fs]

attachBottomRightCorner
  self performBlockedUpdate: [| fs |
    (fs := self visualPolicy visualPartFramingSpecTranslateBy: 0@0)
      leftEdge: (fs leftEdge
        attachment: AbtAttachmentsConstants::XmATTACHSELFOPPOSITE
        currentView: self part);
      rightEdge: (fs rightEdge attachment: XmATTACHFORM currentView: self part);
      topEdge: (fs topEdge
        attachment: AbtAttachmentsConstants::XmATTACHSELFOPPOSITE
        currentView: self part);
      bottomEdge: (fs bottomEdge attachment: XmATTACHFORM currentView: self part).
    self frameVisualPart: fs]
```

# Attachments - Sample Code 3

- Add the following methods to AbtCwEditPart (continued)

```
attachBottomLeftCorner
  self performBlockedUpdate: [| fs |
    (fs := self visualPolicy visualPartFramingSpecTranslateBy: 0@0)
      leftEdge: (fs leftEdge attachment: XmATTACHFORM currentView: self part);
      rightEdge: (fs rightEdge
        attachment: AbtAttachmentsConstants::XmATTACHSELFOPPOSITE
        currentView: self part);
      topEdge: (fs topEdge
        attachment: AbtAttachmentsConstants::XmATTACHSELFOPPOSITE
        currentView: self part);
      bottomEdge: (fs bottomEdge attachment: XmATTACHFORM currentView: self part).
    self frameVisualPart: fs]

attachTopBottomRightSides
  self performBlockedUpdate: [| fs |
    (fs := self visualPolicy visualPartFramingSpecTranslateBy: 0@0)
      leftEdge: (fs leftEdge
        attachment: AbtAttachmentsConstants::XmATTACHSELFOPPOSITE
        currentView: self part);
      rightEdge: (fs rightEdge attachment: XmATTACHFORM currentView: self part);
      topEdge: (fs topEdge attachment: XmATTACHFORM currentView: self part);
      bottomEdge: (fs bottomEdge attachment: XmATTACHFORM currentView: self part).
    self frameVisualPart: fs]
```

# Attachments - Sample Code 4

- Add the following methods to AbtCwEditPart (continued)

```
addAttachmentItemsToPopUpMenu: aPopUpMenu
    | cascadeMenu |
    cascadeMenu := aPopUpMenu
        createPulldownMenu: 'Set Attachments'
        argBlock: nil.
    (aPopUpMenu
        createCascadeButton: 'Set Attachments'
        argBlock: [:w | w subMenuId: cascadeMenu])
        manageChild.
    (cascadeMenu
        createToggleButton: 'All Sides' argBlock: nil)
        addCallback: XmNvalueChangedCallback
            receiver: [:editPart :clientDate :callData |
                self attachAllSides]
            selector: #value:value:value:
            clientData: nil;
        manageChild.
    ...
```

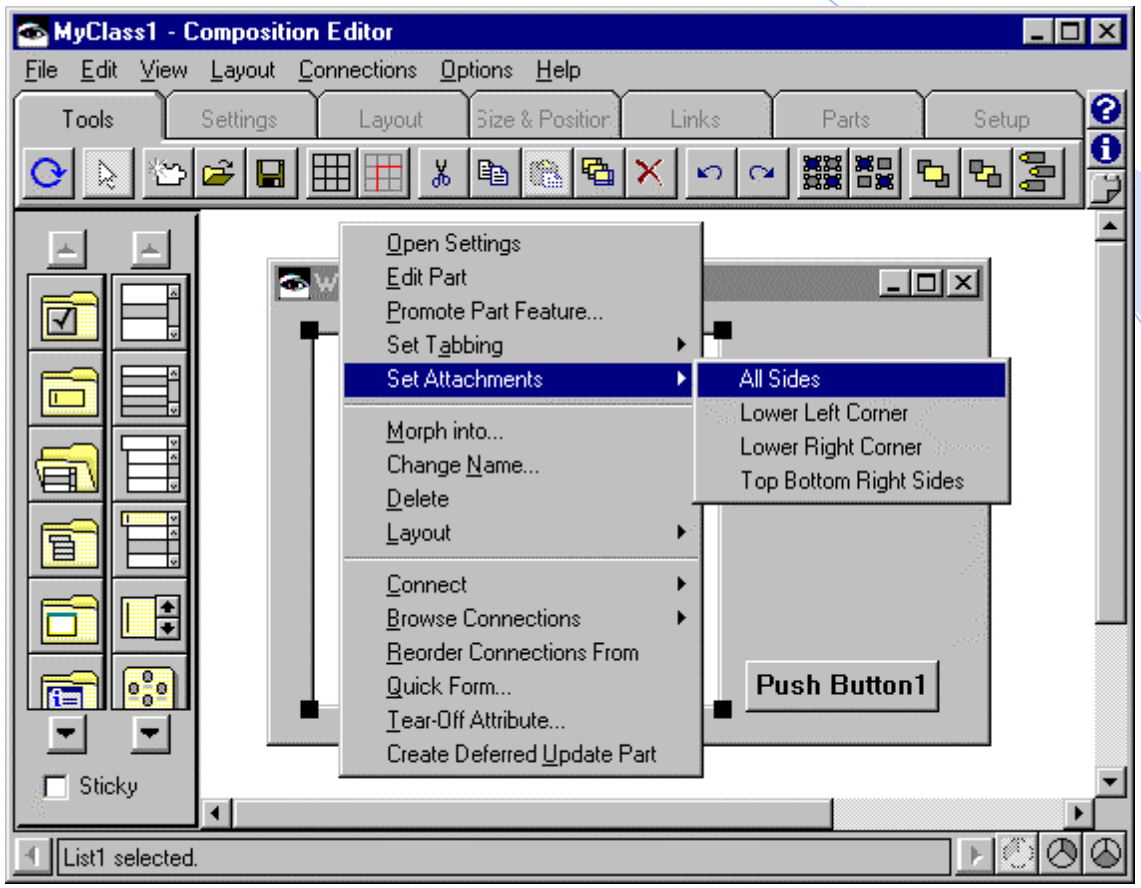
# Attachments - Sample Code 5

- The # addAttachmentItemsToPopUpMenu: method continued

```
...
(cascadeMenu
    createToggleButton: 'Lower Left Corner' argBlock: nil)
    addCallback: XmNvalueChangedCallback
        receiver: [:editPart :clientDate :callData |
            self attachBottomLeftCorner]
        selector: #value:value:value:
        clientData: nil;
        manageChild.
(cascadeMenu
    createToggleButton: 'Lower Right Corner' argBlock: nil)
    addCallback: XmNvalueChangedCallback
        receiver: [:editPart :clientDate :callData |
            self attachBottomRightCorner]
        selector: #value:value:value:
        clientData: nil;
    manageChild.
(cascadeMenu
    createToggleButton: 'Top Bottom Right Sides' argBlock: nil)
    addCallback: XmNvalueChangedCallback
        receiver: [:editPart :clientDate :callData |
            self attachTopBottomRightSides]
        selector: #value:value:value:
        clientData: nil;
    manageChild.
```

# Attachments - New Menu

- Now we can set attachments like this



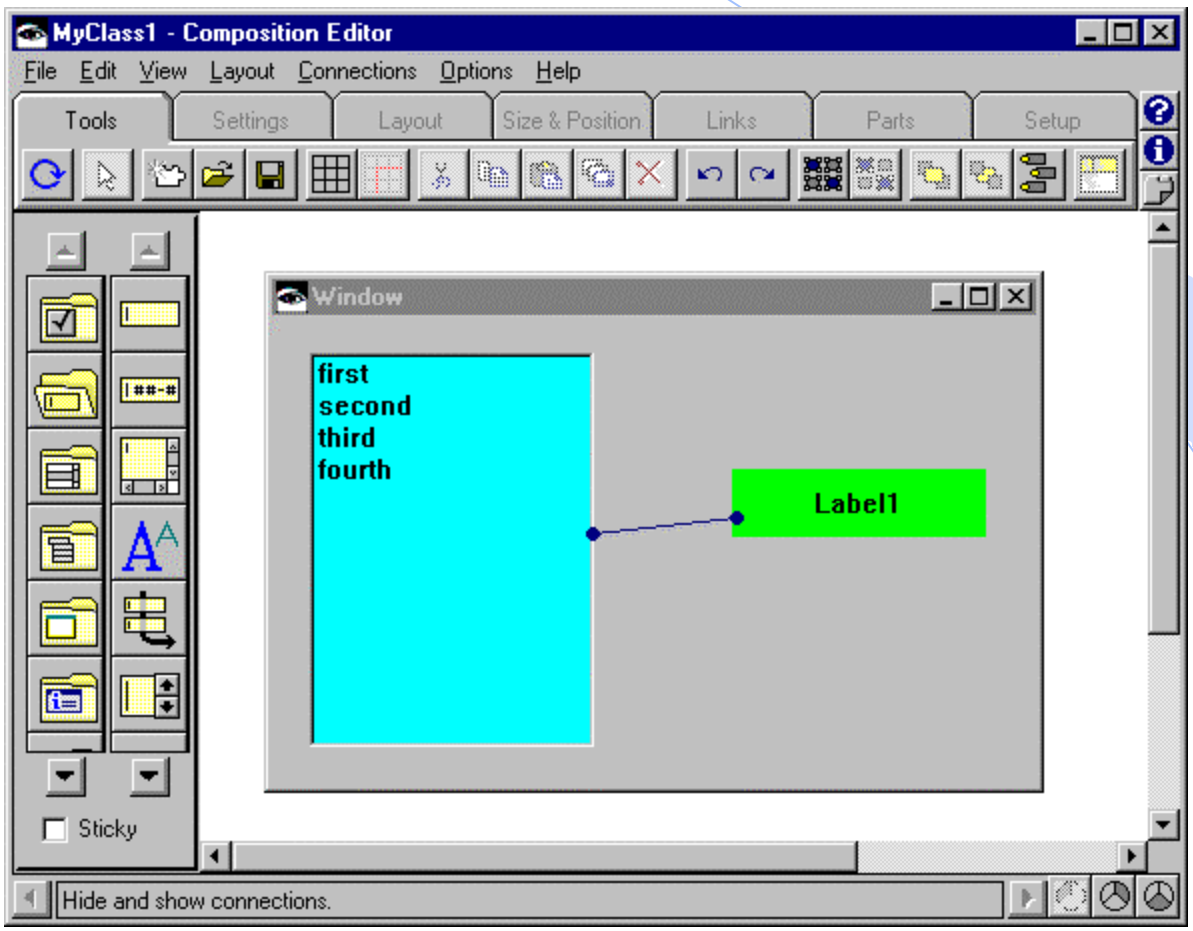
# Morphing

- What is “morphing”?
  - Replace any widget in the Composition Editor with another
  - Maintain any common attributes
  - Maintain any links that still make sense
- VisualAge has a built-in framework that is used in only one place!
  - Morphing obsolete AbtNotebookView to AbtPortablePMNotebookView
  - Very easy to extend
  - Just add a #abtIsomorphicClasses class method to any AbtPart subclass
    - Answer a collection of symbols representing the classes that are valid replacements
    - Examples:

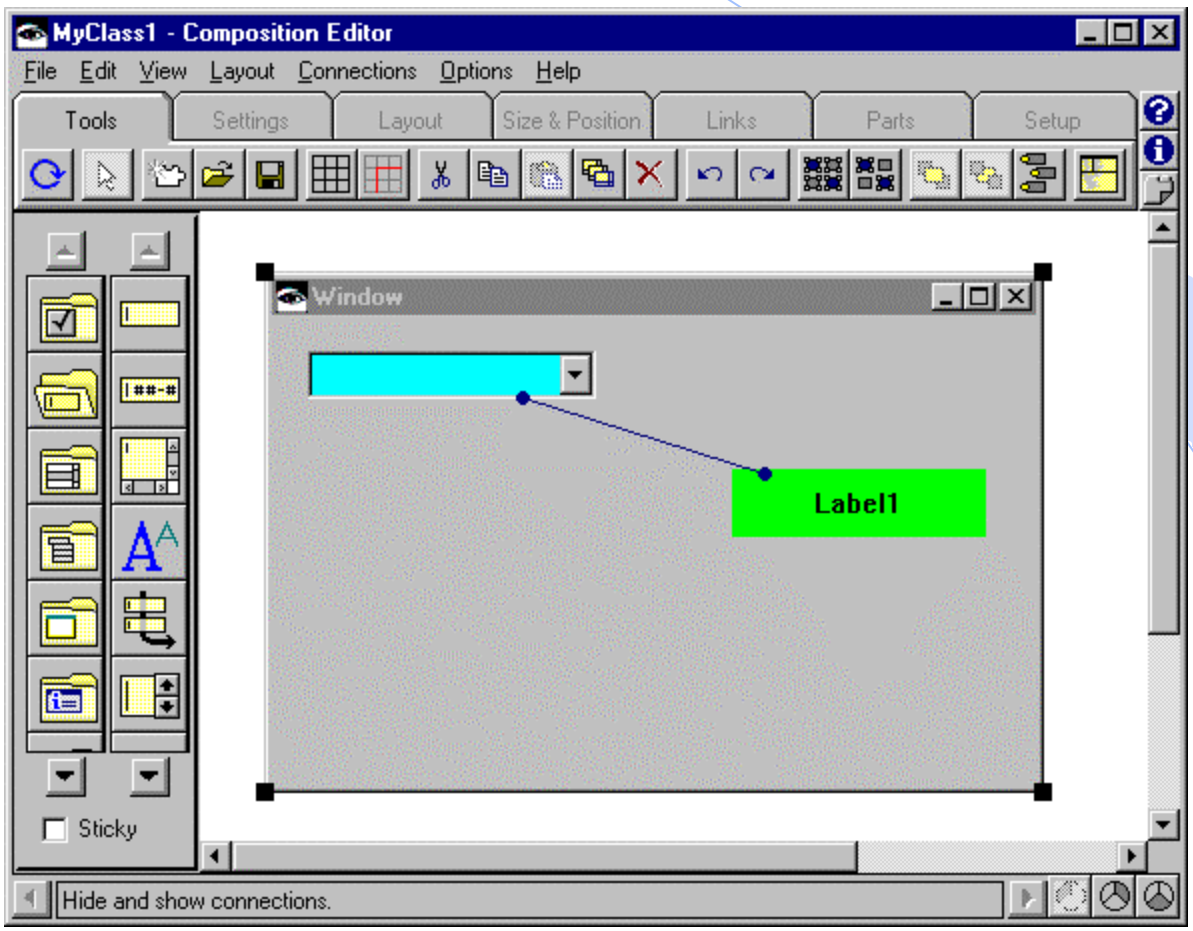
```
AbtListView class>>abtIsomorphicClasses
    ^#(#AbtDropDownListComboBox #AbtComboBoxView
        #AbtContainerDetailsView #AbtMultipleSelectListView
        #AbtSpinButtonView)

AbtMultipleSelectListView class>>abtIsomorphicClasses
    ^#(#AbtDropDownListComboBox #AbtComboBoxView
        #AbtContainerDetailsView #AbtListView
        #AbtSpinButtonView)
```

# Morphing Example - Before

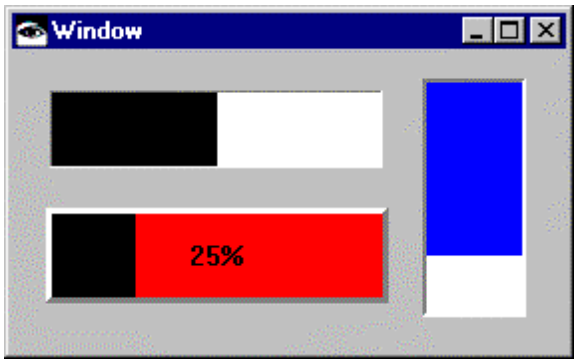


# Morphing Example - After



# Custom Visual Part Development

- General Process
  - Subclass AbtPrimitiveView
  - Define Accessors
  - Define Helper Methods
  - Define Properties
  - Edit-time Extensions
  - Add to Tool Palette
- Example
  - VisualAge contains a nice progress bar widget called EwProgressBar
  - EwProgressBar is a CwWidget-layer component
  - We'll make an AbtPart layer component out of it



# Subclass AbtPrimitiveView

- Create MyAbtProgressBarView as a subclass of AbtPrimitiveView
- Specify which CwWidget subclass to use at the core of the part by adding a #cwWidgetClass class method to MyAbtProgressBarView

```
cwWidgetClass
      ^EwProgressBar
```

- Add instance variables to hold the various attributes needed by the part
  - shadowType
  - shadowWidth
  - orientation
  - direction
  - fractionComplete
  - showPercentage
  - imageColor
  - graphicsDescriptor
  - ribbonGraphicsDescriptor

# Define Accessors

- Create accessor methods for the various properties

```
direction
	direction == nil ifTrue: [^XmFORWARD].
	^direction

direction: anInt
	direction := anInt.
	widget notNil ifTrue: [widget direction: anInt].
	self signalEvent: #directionChanged with: anInt.

fractionComplete
	fractionComplete == nil ifTrue: [^0].
	^fractionComplete

fractionComplete: anInt
	fractionComplete := anInt.
	widget notNil ifTrue: [widget fractionComplete: anInt / 100].
	self signalEvent: #fractionCompleteChanged with: anInt.
```

# Define Accessors - 2

- Create accessor methods for the various properties (continued)

```
orientation
	orientation == nil ifTrue: [^XmHORIZONTAL].
	^orientation

orientation: anInt
	orientation := anInt.
	widget notNil ifTrue: [widget orientation: anInt].
	self signalEvent: #orientationChanged with: anInt.

shadowType
	shadowType == nil ifTrue: [^XmSHADOWIN].
	^shadowType

shadowType: anInt
	shadowType := anInt.
	widget notNil ifTrue: [widget shadowType: anInt].
	self signalEvent: #shadowTypeChanged with: anInt.
```

# Define Accessors - 3

- Create accessor methods for the various properties (continued)

```
shadowWidth
	shadowWidth == nil ifTrue: [^1].
	^shadowWidth

shadowWidth: anInt
	shadowWidth := anInt.
	widget notNil ifTrue: [widget shadowWidth: anInt].
	self signalEvent: #shadowWidthChanged with: anInt.

showPercentage
	showPercentage == nil ifTrue: [^false].
	^showPercentage

showPercentage: aBoolean
	showPercentage := aBoolean.
	widget notNil ifTrue: [widget showPercentage: aBoolean].
	self signalEvent: #showPercentageChanged with: aBoolean.
```

# Define Accessors - 3

- Create accessor methods for the various properties (continued)

```
imageColor
    ^imageColor

imageColor: aString
    (aString = imageColor
        or: [aString noNil and: [aString isEmpty]])
        ifTrue: [^nil].
    imageColor := aString.
    widget notNil
        ifTrue: [widget imageColor: (self asRgb: imageColor)].
    self signalEvent: #imageColorChanged with: imageColor.
```

- The #asRgb: method converts color strings (e.g., “red”) into instances of CgRGBColor (e.g., CgRGBColor red: 65536 green: 0 blue: 0)

# Define Accessors - 5

- Create accessor methods for the various properties (continued)

```
graphicsDescriptor
	^graphicsDescriptor

graphicsDescriptor: aGraphicsDescriptor
	graphicsDescriptor := aGraphicsDescriptor.
	widget notNil ifTrue: [self updateGraphic: widget].
	self signalEvent: #graphicsDescriptorChanged
		with: aGraphicsDescriptor.

ribbonGraphicsDescriptor
	^ribbonGraphicsDescriptor

ribbonGraphicsDescriptor: aGraphicsDescriptor
	ribbonGraphicsDescriptor := aGraphicsDescriptor.
	widget notNil ifTrue: [self updateGraphic: widget].
	self signalEvent: #ribbonGraphicsDescriptorChanged
		with: aGraphicsDescriptor.
```

# Helper Methods

- Create helper methods for handling graphic descriptors

```
calcGraphicLabelType
    ^((graphicsDescriptor isNil or: [graphicsDescriptor isIconDescriptor])
        and: [ribbonGraphicsDescriptor isNil
        or: [ribbonGraphicsDescriptor isIconDescriptor]])
        ifTrue: [XmICON]
        ifFalse: [XmPIXMAP]

updateGraphic: aWidget
    self calcGraphicLabelType == XmICON
        ifTrue: [
            graphicsDescriptor isNil
                ifFalse: [aWidget image: graphicsDescriptor icon].
            ribbonGraphicsDescriptor isNil
                ifFalse: [aWidget ribbonImage: ribbonGraphicsDescriptor icon]]
        ifFalse: [
            graphicsDescriptor isNil
                ifFalse: [aWidget image: graphicsDescriptor pixmap].
            ribbonGraphicsDescriptor isNil
                ifFalse: [
                    aWidget ribbonImage: ribbonGraphicsDescriptor pixmap]].
```

# #widgetCreationArgBlock Method

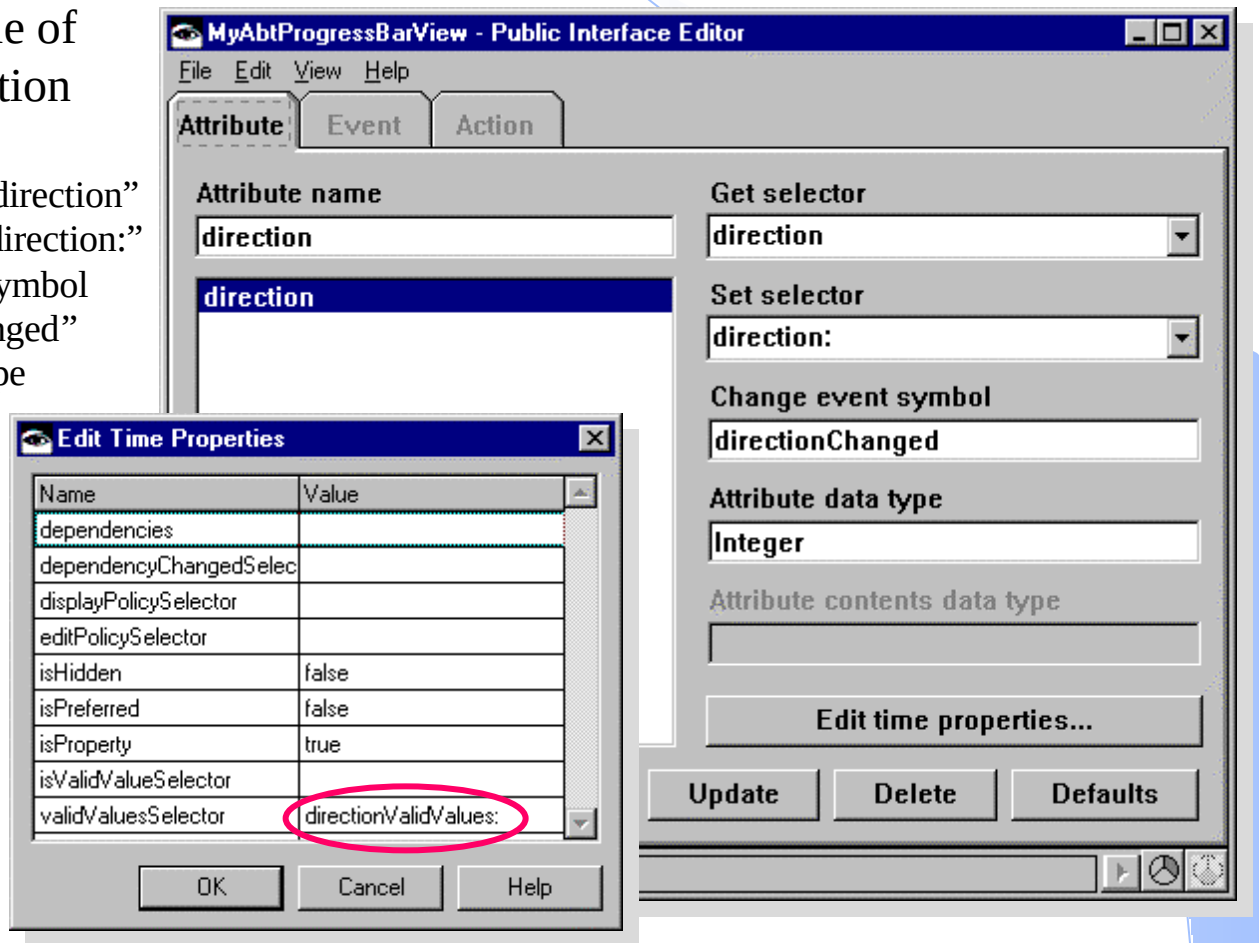
- Create #widgetCreationArgBlock method

```
widgetCreationArgBlock
    ^[:w | super widgetCreationArgBlock value: w.
        direction == nil ifFalse: [w direction: direction].
        orientation == nil ifFalse: [w orientation: orientation].
        shadowType == nil ifFalse: [w shadowType: shadowType].
        shadowWidth == nil ifFalse: [w shadowWidth: shadowWidth].
        fractionComplete == nil
            ifFalse: [w fractionComplete: fractionComplete / 100].
        showPercentage == nil
            ifFalse: [w showPercentage: showPercentage].
        (graphicsDescriptor == nil
            and: [ribbonGraphicsDescriptor == nil])
            ifFalse: [self updateGraphic: w]].
```

- Used during during the creation of the CwWidget-layer component ("w" in the above is an instance of EwProgressBar)

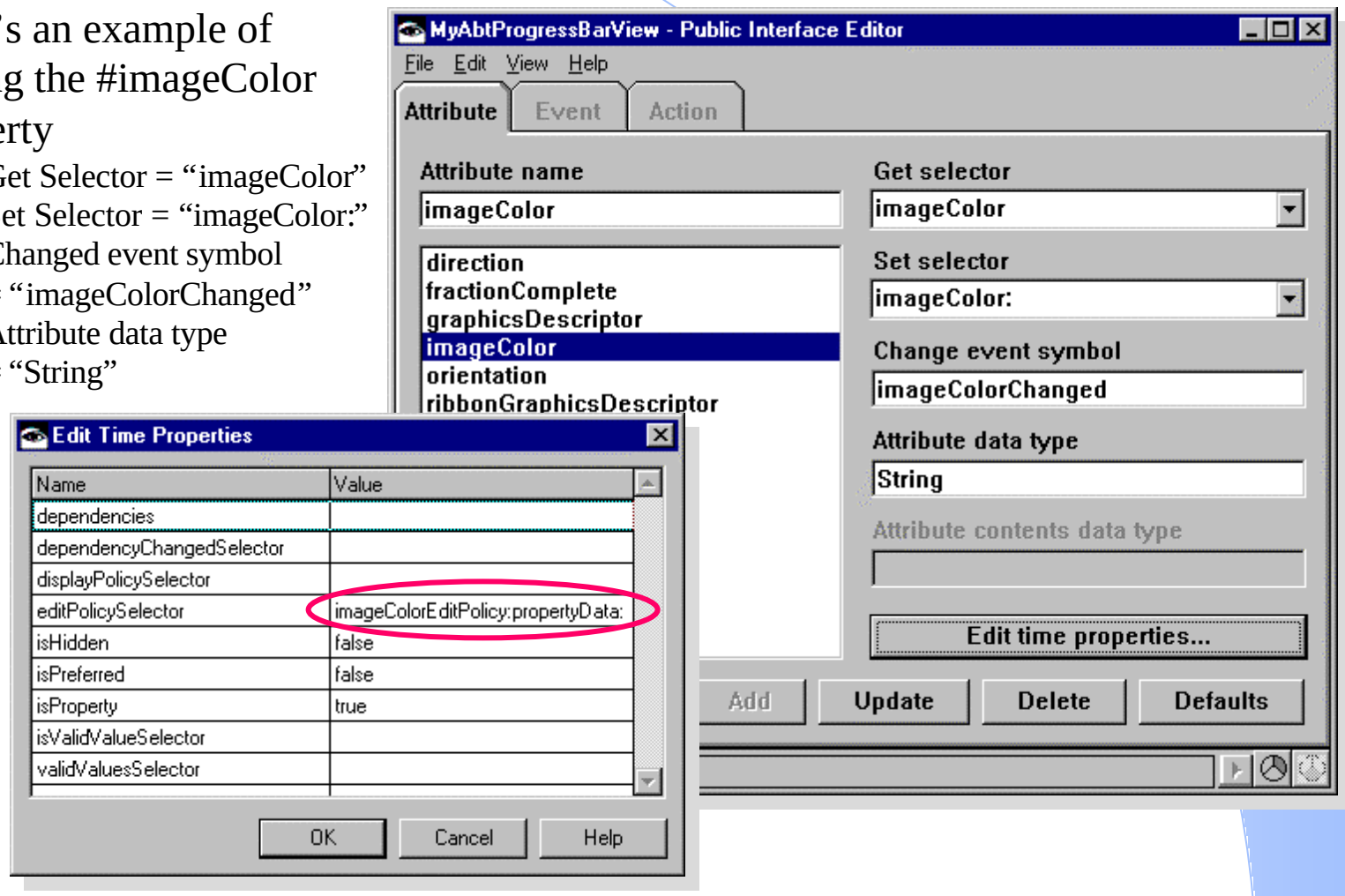
# Define Properties

- Next we open the Public Interface Editor to define all of the properties
- Here's an example of adding the #direction property
  - Get Selector = "direction"
  - Set Selector = "direction:"
  - Changed event symbol = "directionChanged"
  - Attribute data type = "Integer"
- Some attributes need special edit-time only attributes
- Only stored in library!!



# Define Properties - 2

- Here's an example of adding the #imageColor property
  - Get Selector = "imageColor"
  - Set Selector = "imageColor:"
  - Changed event symbol = "imageColorChanged"
  - Attribute data type = "String"



# Edit-time Extensions

- Define edit-time property methods (these provide the values for any attributes with a drop-down selection list)

```
directionValidValues: aPartPropertyData
     ^Dictionary new
               at: 'XmFORWARD' put: XmFORWARD;
               at: 'XmREVERSE' put: XmREVERSE;
               yourself

orientationValidValues: aPartPropertyData
     ^Dictionary new
               at: 'XmHORIZONTAL' put: XmHORIZONTAL;
               at: 'XmVERTICAL' put: XmVERTICAL;
               yourself

shadowTypeValidValues: aPartPropertyData
     ^Dictionary new
               at: 'XmSHADOWNONE' put: XmSHADOWNONE;
               at: 'XmSHADOWIN' put: XmSHADOWIN;
               at: 'XmSHADOWOUT' put: XmSHADOWOUT;
               yourself
```

# Edit-time Extensions - 2

- Define edit-time edit policy methods (these set up the editors for any special properties)

```
imageColorEditPolicy: initialValue propertyData: aPartPropertyData
    ^AbtEwObjectPrompterEditPolicy new
        editable: true;
        value: initialValue;
        prompter: AbtColorNamePrompter new;
        yourself
```

# Edit-time Extensions - 3

- Define miscellaneous class-side edit methods
  - Answer the part's name default size in the Composition Editor

```
defaultEditSize
    ^160 @ 20
```

  - Answer the part's name to be displayed in the status area of the Composition Editor

```
displayName
    ^'Progress Bar'
```

  - Return the descriptor for the icon representing the class

```
abtInstanceGraphicsDescriptor
    ^AbtIconDescriptor new
        moduleName: self abtGraphicsModuleName;
        id: 360
```

  - Magic methods needed to make the part show up at the right size

```
attachmentSpecAt: point
    ^self attachmentSpecFromRect:
        (point extent: self defaultEditSize)

positionSpecAt: point
    ^self positionSpecFromRect:
        (point extent: self defaultEditSize)
```

# Add to Tool Palette

- Add class methods to MyApplication to register our new part to the part palette
  - Answer the list of parts

```
abtPaletteParts
    ^#(MyAbtProgressBarView)
```

  - Answer the name of the part category (new or existing)

```
abtPaletteCategoryName
    ^'Progress Bars'
```

  - Answer the category icons (if new category)

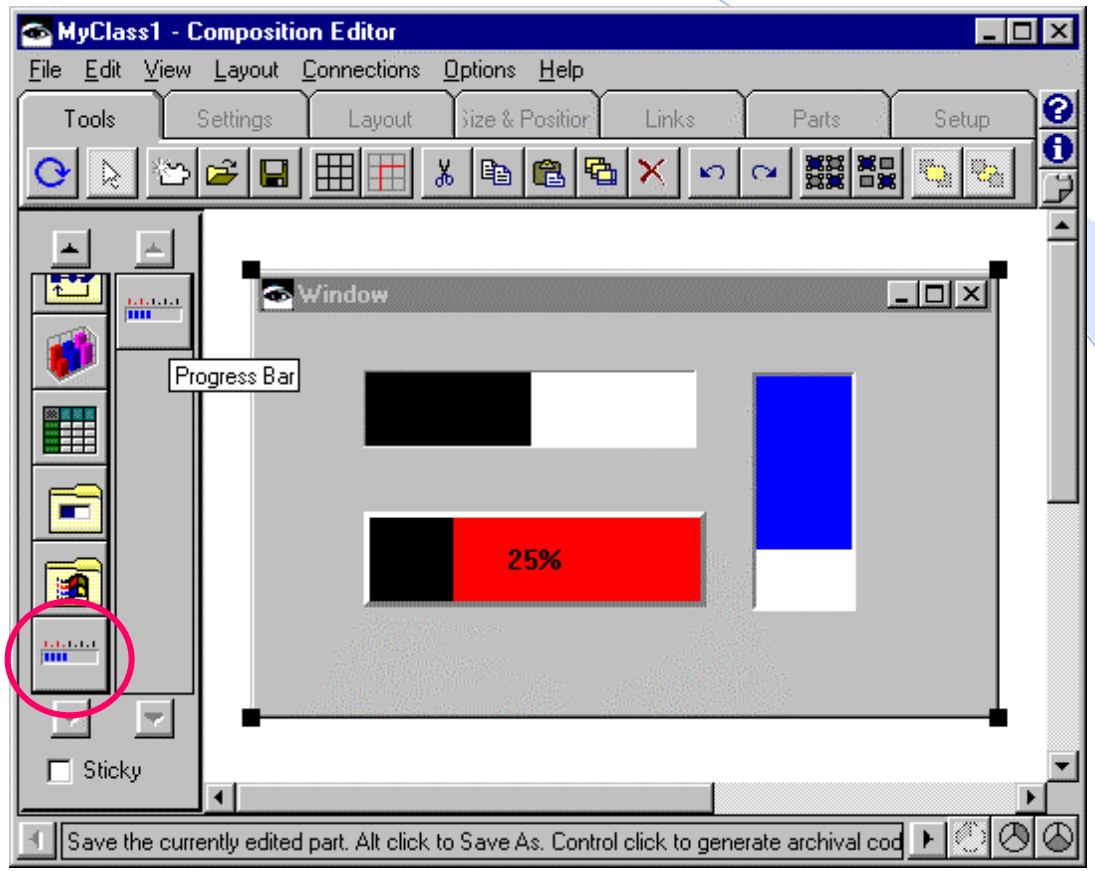
```
abtPaletteCategoryGraphicsDescriptor
    ^AbtIconDescriptor new
        moduleName: self abtGraphicsModuleName;
        id: 360

abtPaletteCategoryOpenGraphicsDescriptor
    ^self abtPaletteCategoryGraphicsDescriptor
```

  - Install and remove our parts when the application is loaded or unloaded

```
loaded
    self abtAddPartsToCatalog
removing
    self abtRemovePartsFromCatalog
```

# MyAbtProgressBarView in Action



# MyAbtSplitBarView Example

- Create MyAbtSplitBarView as a subclass of AbtPrimitiveView
- Specific which CwWidget subclass to use at the core of the part by adding a #cwWidgetClass class method to MyAbtProgressBarView

```
cwWidgetClass
	^CwSash
```

- Add instance variable to hold the various attributes needed by the part
  - orientation
  - topLimitWidget, bottomLimitWidget, leftLimitWidget, rightLimitWidget
- Create accessor methods for the various properties

```
orientation
	orientation == nil ifTrue: [^XmHORIZONTAL].
	^orientation

orientation: anInt
	orientation := anInt.
	widget notNil ifTrue: [widget orientation: anInt].
	self signalEvent: #orientationChanged with: anInt.
```

# MyAbtSplitBarView Example - 2

- Create accessor methods for the various properties (continued)

```
topLimitWidget
	^topLimitWidget

topLimitWidget: anAbtBasicView
	topLimitWidget := anAbtBasicView.
	widget notNil
	    ifTrue: [widget topLimitWidget: anAbtBasicView widget].
	self signalEvent: #topLimitWidgetChanged
	    with: anAbtBasicView.

bottomLimitWidget
	^bottomLimitWidget

bottomLimitWidget: anAbtBasicView
	bottomLimitWidget := anAbtBasicView.
	widget notNil
	    ifTrue: [widget bottomLimitWidget: anAbtBasicView widget].
	self signalEvent: #bottomLimitWidgetChanged
	    with: anAbtBasicView.
```

# MyAbtSplitBarView Example - 3

- Create accessor methods for the various properties (continued)

```
leftLimitWidget
	^leftLimitWidget

leftLimitWidget: anAbtBasicView
	leftLimitWidget := anAbtBasicView.
	widget notNil
	    ifTrue: [widget leftLimitWidget: anAbtBasicView widget].
	self signalEvent: #leftLimitWidgetChanged
	    with: anAbtBasicView.

rightLimitWidget
	^rightLimitWidget

rightLimitWidget: anAbtBasicView
	rightLimitWidget := anAbtBasicView.
	widget notNil
	    ifTrue: [widget rightLimitWidget: anAbtBasicView widget].
	self signalEvent: #rightLimitWidgetChanged
	    with: anAbtBasicView.
```

# MyAbtSplitBarView Example - 4

- Create #widgetCreationArgBlock method

```
widgetCreationArgBlock
    ^[:w | super widgetCreationArgBlock value: w.
         w orientation: orientation.
         topLimitWidget == nil ifFalse: [
             w topLimitWidget: topLimitWidget widget].
         leftLimitWidget == nil ifFalse: [
             w leftLimitWidget: leftLimitWidget widget].
         rightLimitWidget == nil ifFalse: [
             w rightLimitWidget: rightLimitWidget widget ].
         bottomLimitWidget == nil ifFalse: [
             w bottomLimitWidget: bottomLimitWidget widget]]
```

- Define edit-time property methods (these provide the values for any attributes with a drop-down selection list)

```
orientationValidValues: aPartPropertyData
     ^Dictionary new
              at: 'XmHORIZONTAL' put: XmHORIZONTAL;
              at: 'XmVERTICAL' put: XmVERTICAL;
              yourself
```

# MyAbtSplitBarView Example - 5

- Define miscellaneous class-side edit methods
  - Answer the part's name default size in the Composition Editor

```
defaultEditSize
    ^200 @ 4
```

  - Answer the part's name to be displayed in the status area of the Composition Editor

```
displayName
    ^'Split Bar'
```

  - Return the descriptor for the icon representing the class

```
abtInstanceGraphicsDescriptor
    ^AbtIconDescriptor new
        moduleName: self abtGraphicsModuleName;
        id: 317
```

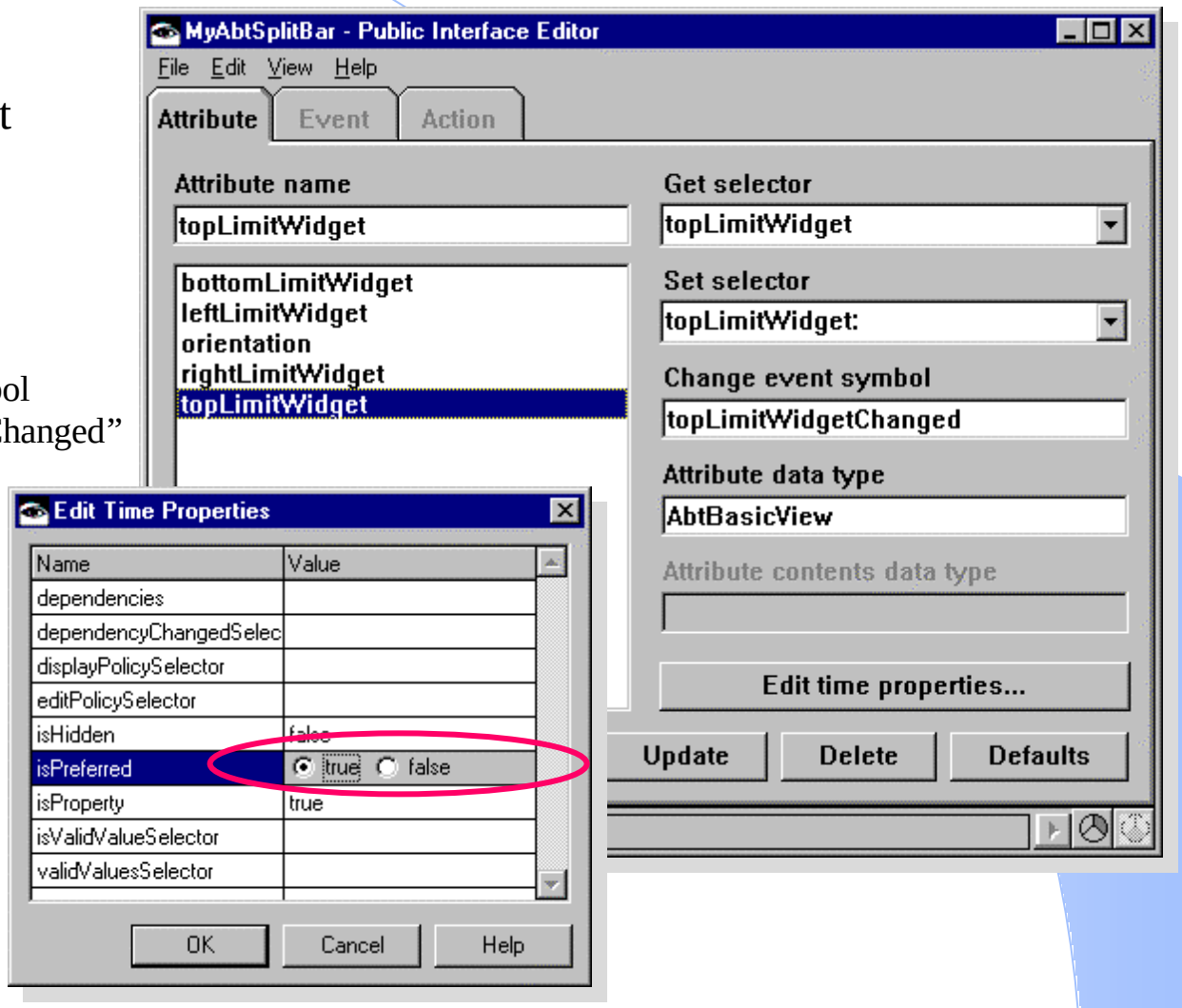
  - Magic methods needed to make the part show up at the right size

```
attachmentSpecAt: point
    ^self attachmentSpecFromRect:
        (point extent: self defaultEditSize)

positionSpecAt: point
    ^self positionSpecFromRect:
        (point extent: self defaultEditSize)
```

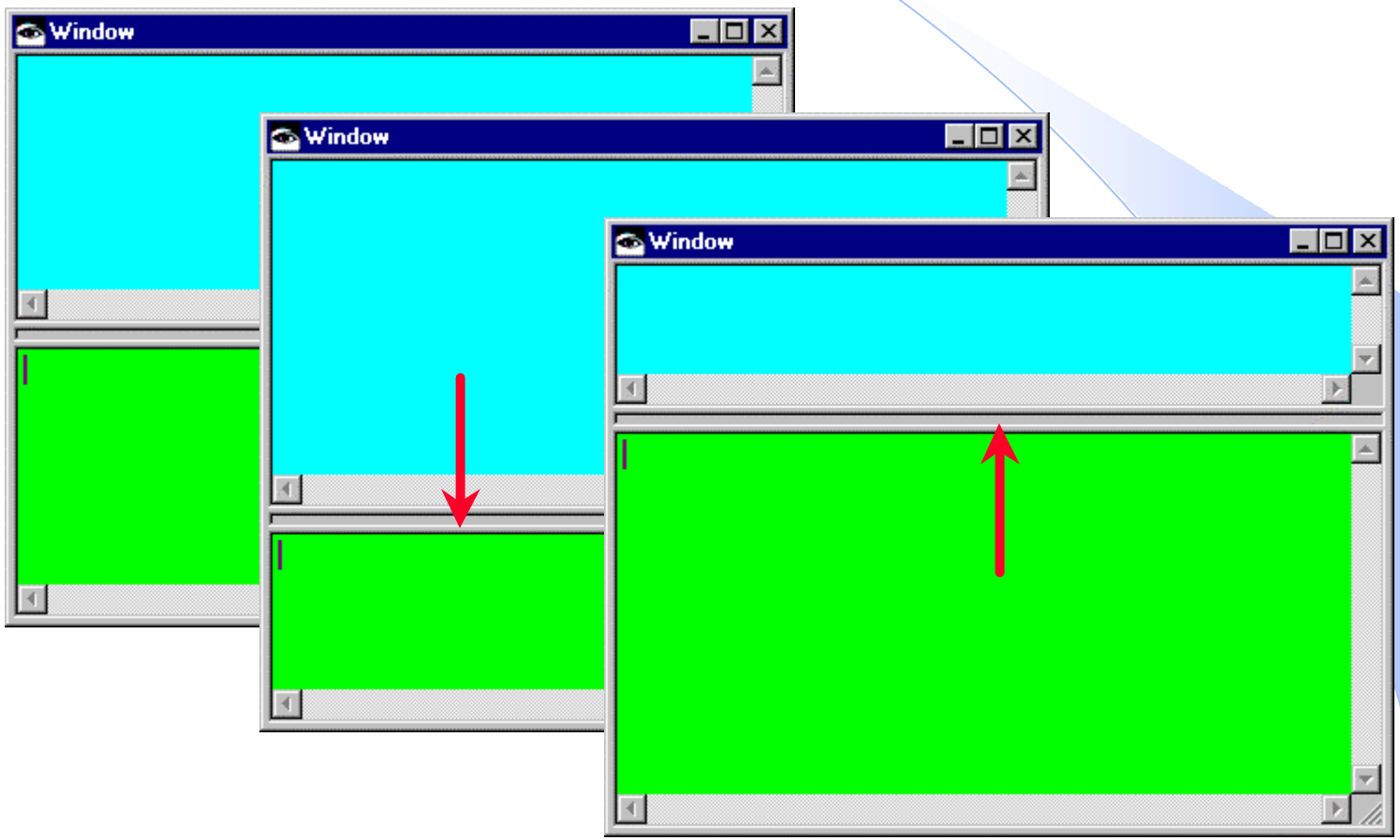
# MyAbtSplitBarView Example - 6

- Example of adding the #topLimitWidget property
  - Get Selector = "topLimitWidget"
  - Set Selector = "topLimitWidget:"
  - Changed event symbol = "topLimitWidgetChanged"
  - Attribute data type = "AbtBasicView"



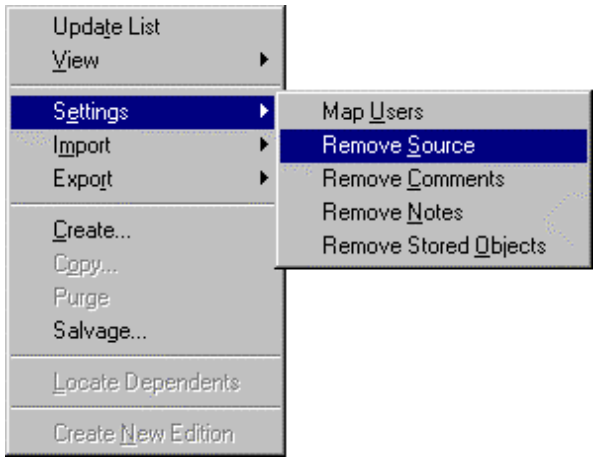
# MyAbtSplitBarView Example - 7

- MyAbtSplitBarView in action



# Complex Configuration Management

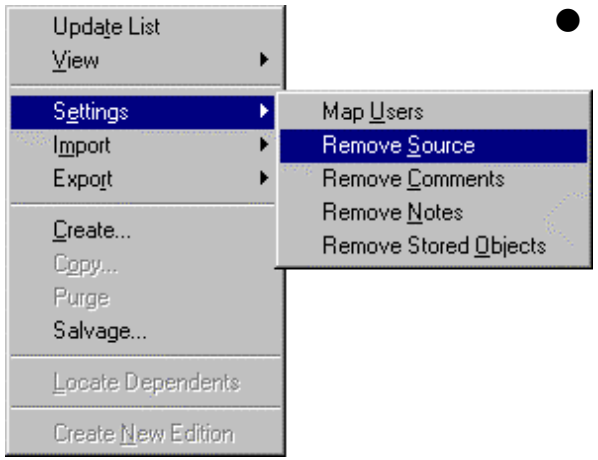
- Hiding Source
- SubApp Configurations
- Version Renaming
- Locating Dependent Configs



# Hiding Source

- Why hide source?
  - Black Box deployment with no “user-serviceable” parts
  - Hide implementation so that a vendor has more freedom to change the guts later on
  - Hide security features (e.g., eval testing / unlocking code)
- Pitfalls
  - Once source is hidden and imported into a manager that DOES have source code, that source code may be wiped out such that developers can no longer view the source to their methods
  - Hiding source for any method that is forced to be recompiled (such as for compile time constants) will break for any VM updates
  - Hiding source should be used SPARINGLY

# Hiding Source - 2




- Mechanics
  - Source is hidden on export to DAT files
  - Source is hidden on an export by export basis (controlled by the Configuration Maps Browser's "Names | Settings | Remove Source" command)
  - What is hidden is stored in an application specific data structure (a Dictionary) that is stored in the library (as an inherited user field)
  - Use the SubApplication class>>removeSourceStructure method to retrieve the current settings
  - Use the SubApplication class>>removeSourceStructure: method to change the current settings
- Date Structure
  - Dictionary of class symbols
  - Values are either
    - "nil" meaning "hide all the source in the class"
    - an Association where the
      - key is either
        - the collection of instance method symbols that should be hidden
        - "nil" to hide all instance methods
      - value is either
        - the collection of class method symbols that should be hidden
        - "nil" to hide all class methods

# Hiding Source - 3

- Example Application: FooBar
  - Class: Foo
    - Class Methods
      - classMethod1
      - classMethod2
    - Instance Methods
      - instanceMethod1
      - instanceMethod2
  - Class: Bar
    - Class Methods
      - classMethod1
      - classMethod2
    - Instance Methods
      - instanceMethod1
      - instanceMethod2

- Hide everything in FooBar

```
FooBar removeSourceStructure:
    (Dictionary new
        at: #Foo put: nil;
        at: #Boo put: nil;
        yourself)
```

- Hide all instance methods in Foo

```
FooBar removeSourceStructure:
    (Dictionary new
        at: #Foo put: (Association key: nil value: #());
        yourself)
```

- Hide all class methods in Bar

```
FooBar removeSourceStructure:
    (Dictionary new
        at: #Bar put: (Association key: #() value: nil);
        yourself)
```

- Hide one class and one instance method in Foo

```
FooBar removeSourceStructure:
    (Dictionary new
        at: #Foo put:
            (Association
                key: #(#instanceMethod1)
                value: #(#classMethod2));
        yourself)
```

# SubApp Configurations

- Why Use?
  - Organize functionality
  - Custom Loading
    - OS-specific
    - Other conditions
- Sample config expressions
  - Load always
    ```
    true
    ```
  - Window only
    ```
    #('WIN32s' 'WIN-NT') includes:
     (System subsystemType: 'CW')
    ```
  - OS/2 only
    ```
    #('PM') includes:
     (System subsystemType: 'CW')
    ```
  - Only if OLE is loaded
    ```
    Smalltalk includesKey:
    #AbtBaseOleApp
    ```
  - Only if Foo is loaded
    ```
    Smalltalk includesKey: #Foo
    ```
- Example
  - MyApp
    - MySubApp1
    - MySubApp2
    - MySubApp3
      …
    - MySubAppN
- Problem
  - Combinatorial explosion
    - 2 subapps = 4 possible configs
    - 3 subapps = 8 possible configs
    - 4 subapps = 16 possible configs
    - Etc.
  - Must be a better way...

# Two-Tier Config Expressions

- Solution to the combinatorial explosion problem
- Rather than
  - MyApp
    - MySubApp1
    - MySubApp2
    - MySubApp3
- Use
  - MyApp
    - MySubApp1Stub
      - MySubApp1
    - MySubApp2Stub
      - MySubApp2
    - MySubApp3Stub
      - MySubApp3
- In first case, MyApp would need up to 8 different *complex* configs to support loading each subapp independently from its siblings
- In the second case, MyApp would need only one config (i.e., “true”) that would load all of its subapps
- Each sub app would then have *simple* configs that only controlled the loading of its single subapp
- This technique can also be used at the config map and application level to solve the problem of context-sensitive prereqs

# Two-Tier Config Expressions Example

- Two-Tier Configs can be used by third-parties to avoid loading collisions
- Example
  - The ubiquitous Object>>asString method
    - Not part of the VisualAge base
    - Supplied by several third parties
    - Common source of conflicts
- Solution: Two-Tier Configs
  - MyApp
    - MyObject_asStringStub
      - MyObject_asStringApp
  - Configuration Expression

```
(Object respondsTo: #asString) not
    or: [(Object>>#asString) application name == #MyApp]
```

# Expression Indicator

- Here's a handy mod which will make it easy for you to tell when a config expression is currently true or not
  - First, implement the following method in EtWindow:

```
expressionIndicatorBlock
    ^[:exp |
        ([Compiler evaluate: exp] when: ExError
            do: [:sig | sig exitWith: nil]) == true
                ifTrue: [EtTools loadedIndicator]
                ifFalse: [EtTools blankLoadedIndicator]]
```

  - Second, modify any #expressionsListWidget method to set the #statusBlock: parameter to "self expressionIndicatorBlock". Here are two:
    - EtApplicationEditionsBrowser>> expressionsListWidget
    - EtConfigurationMapsBrowser>> expressionsListWidget

# Version Renaming

- Why rename versions?
  - Consistency
  - Baselining apps and classes for delivery
  - Correcting naming mistakes
- Why isn't this dangerous?
  - The ENVY library only cares about time stamps
  - APIs exist to change version names after they have been set
  - These APIs have remained consistent for many years
  - IBM/OTI uses this technique to baseline VisualAge releases
  - All version sorting is done by timestamp. Version names are cosmetic only

# Version Renaming - Applications

- Pick a version name and select the applications to modify
- Iterate over the application list
- For each application, compare its version name to the new desired name (no point in changing the name if it isn't necessary)
- For each application that needs changing, update the edition record

```
| versionName applications |
versionName := <New Version Name>.
applications := Array with: <App1> with: <App2>.
applications do: [:application |
    application timeStamp versionName = versionName
        ifFalse: [
            application updateEdition: [:editionRecord |
                editionRecord
                    setVersionName: versionName;
                    insert]]].
```

# Version Renaming - Classes

- Pick a version name, an application and a set of classes to modify
- Iterate over the class list
- For each class, compare its version name to the new desired name
- For each class that needs changing, update the edition record

```
| versionName application classes |
versionName := <New Version Name>.
application := <Application>.
classes := Array with: <Class1> with: <Class2>.
classes do: [:class |
    timeStamp := class timeStampIn: application.
    timeStamp versionName = versionName
        ifFalse: [
            timeStamp versionName: versionName.
            class updateIn: application with: [:editionsRecord |
                | entry oldLength |
                entry := editionsRecord currentEntry.
                oldLength := entry versionName size.
                entry
                    replaceElement: 2 with: versionName;
                    length: entry length - oldLength + versionName size;
                    yourself]]].
```

# Version Renaming - Config Maps

- Pick a version name and select the configuration map to modify
- Find the most recent edition of the config map
- Update the edition record of the config map edition with the new version name

```
| versionName configMapName configMapEdition |
versionName := <New Version Name>.
configMapName := <Config Map Name>.
configMapEdition := (EmConfigurationMap
     editionsFor: configMapName) first.
configMapEdition
     relocateRecordWith: [:editionRecord |
               editionRecord
                        replaceElement: 2 with: versionName;
                        insert].
```

# Locating Dependent Configs for an (Sub)Application

- Get the name of the root application
- Scan through all Config Map names in the system
- For each configuration, find the first (most recent edition)
- Check to see whether its application names include the target

```
| appName dependentConfigs |
appName := <Application> rootApplication name asString.
dependentConfigs := EmConfigurationMap configurationMapNames
     select: [:mapName | | editions |
              editions := EmConfigurationMap editionsFor: mapName.
              editions first applicationNames
                       includes: appName].
^dependentConfigs
```

# Locating Dependent Configs for a Config Map (Direct)

- Specify the name of the configuration map
- Scan through all Config Map names in the system
- For each configuration, find the first (most recent edition)
- Check to see whether its required maps names include the target

```
| configName dependentConfigs |
configName := <Configuration Map Name>.
dependentConfigs := EmConfigurationMap configurationMapNames
     select: [:mapName | | map |
              map := (EmConfigurationMap editionsFor: mapName) first.
              (map allPossibleRequiredMaps
                       detect: [:mp | mp name = configName]
                       ifNone: []) notNil].
^dependentConfigs
```

# Locating Dependent Configs for a Config Map (Indirect)

- Collect the names of all of the application names contained by the map
- Scan through all Config Map names in the system
- For each configuration, find the first (most recent edition)
- Check to see whether its application names names include the all of the application names in the target

```
| configName applicationNames dependentConfigs |
configName := <Configuration Map Name>.
applicationNames := (EmConfigurationMap
     editionsFor: configName) first applicationNames.
dependentConfigs := EmConfigurationMap configurationMapNames
     select: [:mapName | | editions names |
               mapName ~= configName and: [
               editions := EmConfigurationMap editionsFor: mapName.
               names := editions first applicationNames.
               applicationNames conform: [:app |
                        names includes: app]]].
^dependentConfigs
```

# Development Tool (Browser) Enhancements

- Extension API
- Subclassing TextSelectionManager
- Hooking KeyPress in Text Widgets
- Enhanced Text Menu

# Extension API

- What is it?
  - Create by Joseph Pelrine
  - Public domain
  - Easy way for multiple vendors (and users) to extend the VisualAge browsers without collision
- How does it work?
  - Overrides the normal #classesMenu (and other menu creation methods) with code that (essentially) looks like this:

```
classesMenu
    | aMenu |
    aMenu := super classesMenu.
    SubApplication currentlyLoaded reverseDo: [:app |
        app addToClassesMenu: aMenu browser: self].
    ^aMenu
```

  - Adds a #addToClassesMenu:browser: method (and siblings) to SubApplication that does nothing
    - First argument is the menu being added to
    - Second argument is the current browser (a source of valuable state information)
  - Other applications override these methods to add in their own menu commands

# Example - Adding All Instances

- Create an application called MyApplication
- Add the following class method to the MyApplication class:

```
addToClassesMenu: aMenu browser: aBrowser
    ^aMenu
        addLine;
        add: #allSelectedClassInstances
            label: 'All ~Instances'
            enable: [aBrowser isOneClassSelected];
        yourself
```

- Add the following method to the EtCodeWindow class:

```
allSelectedClassInstances
     self selectedClass allInstances inspect
```

- All of the Classes menus in all of the browsers should now have an "All Instances" method which will automatically enable/disable whenever a class is selected or not

# Using Progress Dialogs

- VisualAge has a nice progress dialog facility you can use for managing long running, interruptible tasks
- Use the EtWindow>> execLongOperation:message:allowCancel:showProgress: method
  - First parameter is a one-argument block of code that will be forked to a background process. The block argument is the dialog itself
  - The “message” parameter is the text displayed in the dialog
  - The “allowCancel” parameter determines whether a Cancel button is available
  - The “showProgress” parameter determines whether a progress bar is displayed
- Several messages can be sent to the block argument (dialog) above
  - #fractionComplete: - set the value shown on the progress bar (a fraction between 0 and 100)
  - #messageString: - sets the message string in the dialog
  - #cancelled - answers a boolean specifying whether the Cancel button was clicked

# Example - Finding Strings

- Modify our #addToClassesMenu:browser: method like this:

```
addToClassesMenu: aMenu browser: aBrowser
    ^aMenu
        addLine;
        add: #allSelectedClassInstances
            label: 'All ~Instances'
            enable: [aBrowser isOneClassSelected];
        add: #findStringInClass
            label: 'Find String In Class'
            enable: [aBrowser isOneClassSelected];
        yourself
```

# Example - Finding Strings - 2

- Add the following method to the EtCodeWindow class:

```
findStringInClass
    | aString found |
    aString := System prompt: 'Methods including string?'.
    (aString isNil or: [aString isEmpty]) ifTrue: [^self].
    self
        execLongOperation: [:dialog |
            found := self
                findString: aString
                inClass: self selectedClass
                dialog: dialog]
        message: 'Gathering methods...'
        allowCancel: true
        showProgress: true.
    found isEmpty
        ifTrue: [System message: 'None found.']
        ifFalse: [
            ((EtTools browser: #highlightingMethods)
                on: (found asSet asSortedCollection: CompiledMethod sortBlock)
                labeled: ('Methods in %1 including %2'
                    bindWith: self selectedClass with: aString printString)
                highlighting: aString)
                    owningImage: System image;
                    open]
```

# Example - Finding Strings - 3

- Also add this method to the EtCodeWindow class:

```
findString: aString inClass: aClass dialog: dialog
    | methods size found cancelled |
    methods := OrderedCollection new.
    aClass methodDictionary do: [:method |
        methods add: method].
    aClass class methodDictionary do: [:method |
        methods add: method].
    size := methods size.
    dialog fractionComplete: 0.
    dialog messageString: 'Found: 0'.
    found := OrderedCollection new.
    cancelled := false.
    methods doWithIndex: [:method :index | | source |
        (cancelled := cancelled or: [dialog cancelled])
            ifFalse: [
                source := method record source.
                (source notNil and: [
                    (source
                        indexOfSubCollection: aString
                        startingAt: 1 ifAbsent: [0]) > 0])
                    ifTrue: [
                        found add: method.
                        dialog messageString: 'Found: ', found size printString].
                dialog fractionComplete: index / size]].
    ^found
```

# Enhancing the Text Selection Manager

- What is the Text Selection Manager?
  - Handles double-click word select
  - Handles finding matching parens and brackets
- What can we do to enhance it?
  - Add double-click line select
  - Watch for special key strokes to insert text or expand abbreviations
- How do we start?
  - Subclass CwSmalltalkTextSelectionManager with MyTextSelectionManager
  - Override the #new method so that we get our version instead:

```
new
    ^MyTextSelectionManager basicNew
```

Very Sneaky

- Override the #doubleClick method like this:

```
doubleClick
    super doubleClick
        ifTrue: [^true].
    self selectLine
        ifTrue: [^true].
    ^false
```

# Enhancing the Text Selection Manager - 2

- Override the #selectWord method like this:

```
selectWord
        | leftPos rightPos |
        leftPos := self findSeparatorLeft.
        rightPos := self findSeparatorRight.
        leftPos == rightPos ifTrue: [^false].
        CwAppContext default asyncExecInUI: [
                self owner setSelection: leftPos @ rightPos].
        ^true
```

- Implement the #selectLine method like this:

```
selectLine
        | leftPos rightPos |
        leftPos := self findLineEndLeft.
        rightPos := self findLineEndRight.
        CwAppContext default asyncExecInUI: [
                self owner setSelection: leftPos @ rightPos].
        ^true
```

# Enhancing the Text Selection Manager - 3

- Implement the #findLineEndLeft method like this:

```
findLineEndLeft
    | findStream lineDelimiter position start |
    findStream := self contentStream.
    lineDelimiter := findStream lineDelimiter.
    (position := self cursorPos) == 0 ifTrue: [^0].
    [position > 0 and: [start isNil]] whileTrue: [
        position := position - 1.
        findStream position: position.
        (lineDelimiter includes: findStream peek)
            ifTrue: [start := position + 1]].
    position := start.
    [findStream atEnd] whileFalse: [
        findStream next isSeparator
            ifFalse: [^findStream position - 1]].
    ^self ownerSize
```

# Enhancing the Text Selection Manager - 4

- Implement the #findLineEndRight method like this:

```
findLineEndRight
    | findStream lineDelimiter position |
    (findStream := self contentStream) position: self cursorPos.
    lineDelimiter := findStream lineDelimiter.
    [findStream atEnd not and: [position isNil]] whileTrue: [
        (lineDelimiter includes: findStream next)
            ifTrue: [position := findStream position - 1]].
    position isNil
        ifTrue: [^self ownerSize].
    [position = 0] whileFalse: [
        position := position - 1.
        findStream position: position.
        findStream peek isSeparator
            ifFalse: [^position + 1]].
    ^0
```

# Hooking KeyPress in Browser Text Widgets

- What else can we do with our new Text Selection Manager?
  - Watch for special key strokes
  - Examples
    - VW goodies - Ctrl+g/f/t
    - Inserting parens, brackets, etc.
    - Expanding Abbreviations
- Let's start with the first one: "VW Goodies"
- How do we do it?
  - Override the CwTextManager class>>for: method

```
for: aCwText
    | manager |
    manager := super for: aCwText.
    aCwText
        addEventHandler: KeyPressMask
        receiver: self
        selector: #keyPress:clientData:callData:
        clientData: manager.
    ^manager
```

# VW Goodies - Ctrl+G/F/T

- VisualWorks implements several keyboard macros
  - Ctrl+G inserts ":="
  - Ctrl+T inserts "ifTrue:"
  - Ctrl+F inserts "ifFalse:"
- Implement the MyTextSelectionManager>> insertString: method

```
insertString: aString
    | pos |
    pos := self owner getInsertionPosition.
    self owner
        setInputFocus;
        replace: pos toPos: pos + 1 value: aString
```

# VW Goodies - Ctrl+G/F/T - 2

- Implement the MyTextSelectionManager>> keyPress:clientData:callData: method

```
keyPress: textWidget clientData: clientData callData: callData
    | ctrl shift |
    ctrl := callData state anyMask: ControlMask.
    shift := callData state anyMask: ShiftMask.
    ctrl & shift
        ifTrue: [
            callData keysym == XKT
                ifTrue: [self insertString: 'ifTrue: ['].
            callData keysym == XKF
                ifTrue: [self insertString: 'ifFalse: ['].
            callData keysym == XKG
                ifTrue: [self insertString: ':= ']].
```

# Inserting Matching Parens

- Implement the MyTextSelectionManager>> parenthesizeSelectedText method

```
parenthesizeSelectedText
    | selectionPosition |
    (selectionPosition := self owner getSelectionPosition) = (0@0)
        ifTrue: [^self].
    self owner
        replace: selectionPosition x
            toPos: selectionPosition y
            value: '(', self owner getSelection, ')';
        setSelection:
            selectionPosition x @ (selectionPosition y + 2);
        setInputFocus
```

# Inserting Matching Parens - 2

- Modify the MyTextSelectionManager>> keyPress:clientData:callData: method

```
keyPress: textWidget clientData: clientData callData: callData
    | ctrl shift |
    ctrl := callData state anyMask: ControlMask.
    shift := callData state anyMask: ShiftMask.
    ctrl & shift
        ifTrue: [
            callData keysym == XKT
                ifTrue: [self insertString: 'ifTrue: ['].
            callData keysym == XKF
                ifTrue: [self insertString: 'ifFalse: ['].
            callData keysym == XKG
                ifTrue: [self insertString: ':=']].
    ctrl
        ifTrue: [
            (callData keysym == XKparenleft
                or: [callData keysym == XK9])
                ifTrue: [^self parenthesizeSelectedText]].
```

# Expanding Abbreviations

- Implement the MyTextSelectionManager>>insertAbbreviation method

```
insertAbbreviation
    | pos start abbrev expansion |
    pos := self owner getInsertionPosition - 1.
    start := self findSeparatorLeftStartingAt: pos.
    abbrev := self owner value copyFrom: start + 1 to: pos.
    expansion := self class abbreviations
        at: abbrev ifAbsent: [^nil].
    self owner
        setInputFocus;
        replace: start toPos: pos + 1 value: expansion
```

- Implement the #findSeparatorLeftStartingAt: method

```
findSeparatorLeftStartingAt: anInteger
    | findStream position |
    findStream := ReadStream on: self owner value.
    position := anInteger.
    [position = 0] whileFalse: [
        position := position - 1.
        findStream position: position.
        findStream peek isAlphaNumeric ifFalse: [^position + 1]].
    ^0
```

# Expanding Abbreviations - 2

- Implement the MyTextSelectionManager class>>abbreviations method

```
abbreviations
    ^Dictionary new
        at: 'int' put: 'isNil ifTrue: []';
        at: 'inf' put: 'isNil ifFalse: []';
        ...
        yourself
```

- Modify the MyTextSelectionManager>> keyPress:clientData:callData: method

```
keyPress: textWidget clientData: clientData callData: callData
    | ctrl shift |
    ctrl := callData state anyMask: ControlMask.
    shift := callData state anyMask: ShiftMask.
    ctrl & shift ifTrue: [...].
    ctrl ifTrue: [...].
    shift
        ifTrue: [
            callData character == CldtConstants::Space
                ifTrue: [^self insertAbbreviation]].
```

# The Joy of Parse Trees

- VisualAge has a very powerful built in parser
- What is a parse tree?
  - A top down, hierarchical representation of a method
  - Ammo for countless browser hacks!
- What can you use it for?
  - Color syntax highlighting
  - Senders and Implementors
  - Spell Checking
  - Limited static analysis

# Parse Tree Example

- Example Method

```
foo
        self doSomething.
        ^self foo: self bar bar: foo.
```

- Parse Tree

```
EsMethod “foo”
    statements:
        EsStatement “self doSomething”
            EsMessageExpression
                receiver: EsVariableWithBinding “self”
                messagePattern: EsUnaryPattern “doSomething”
        EsStatement “self foo: self bar bar: foo”
            EsMessageExpression
                receiver: EsVariableWithBinding “self”
                messagePattern: EsKeywordPattern “foo: self bar bar: foo”
                    selector: #(#foo: #bar:)
                    arguments:
                        EsMessageExpression “self bar”
                            receiver: EsVariableWithBinding “self”
                            messagePattern: EsUnaryPattern “bar”
                        EsVariableWithBinding “foo”.
```

# Creating a Parse Tree

- The EsCompiler>>parse:forEvaluation:environment:errorHandler: method answers an EsComplilationResult that holds onto a parse tree
- The “forEvaluation” parameter should be false for a method and true for a DoIt.
- The “environment” parameter provides a default namespace that the compiler can use to resolve the variables
- The “errorHandler” parameter is set to an EsSilentErrorHandler (we don’t care about errors)

```
parseTreeFor: aString
    ^(Compiler
        parse: aString
        forEvaluation: false
        environment: (EsNameEnvironment new
            environment: Smalltalk;
            sourceClass: Object)
        errorHandler: EsSilentErrorHandler new) parseTree
```

# Find the Selector at the Cursor

- Get the index of the cursor in the browser text widget
- Generate the parse tree for the text in the browser
- Loop through all of the parse tree nodes looking for the node containing the cursor index
- Answer the selector held by the parse node or nil if the parse node does not represent a selector (e.g., a global, a literal, self, super, etc.)

```
EtWindow>>selectorAtCursor
    | textWidget index parseTree |
    index := (textWidget := self targetTextWidget) cursorPosition.
    (parseTree := self parseTreeFor: textWidget getString) notNil
        ifTrue: [| targetNode |
            parseTree allNodesDo: [:node |
               (node sourceStart - 1 <= index and: [node sourceEnd > index])
                   ifTrue: [targetNode := node]].
            (targetNode notNil and: [targetNode selector notNil])
                ifTrue: [^targetNode selector]].
    ^nil
EsParseNode>>selector
    ^nil
```

# Sender & Implementors

- Senders
  - Find the selector at the cursor
  - Ask the system for all senders of that method

```
EtWindow>>senders
    | symbol |
    (symbol := self selectorAtCursor) isNil
        ifFalse: [
            self owningImage allMethodsSending: symbol]
```

- Implementors
  - Find the selector at the cursor
  - Ask the system for all methods by that name

```
EtWindow>>implementors
    | symbol |
    (symbol := self selectorAtCursor) isNil
        ifFalse: [
            self owningImage allMethodsNamed: symbol]
```

# Enhancing the Popup Text Menu

- Use the Extension API to enhance the EtWindow>>defaultTextMenu method
- Add the following class method to the MyApplication class to add new “Senders” and “Implementors” items

```
addToDefaultTextMenu: aMenu browser: aBrowser
    ^aMenu
        add: #senders
            label: 'Senders'
            enable: true
            after: #menuEditFileIn;
        add: #implementors
            label: 'Implementors'
            enable: true
            after: #senders;
        addLineAfter: #implementors;
        yourself
```

# VisualAge Resources

- IBM
  - Smalltalk Home Page
    http://www.software.ibm.com/ad/smalltalk/
  - Support Page
    http://www.software.ibm.com/ad/smalltalk/support/
  - Tips page
    http://www.software.ibm.com/ad/smalltalk/support/tips55.html
  - Add-on products
    http://www.software.ibm.com/ad/visage/rc/rcst5.html
  - Newsgroup
    news://news.software.ibm.com/ibm.software.vasmalltalk
  - FTP patches
    ftp://ps.boulder.ibm.com/ps/products/visualage/fixes/
  - Download VAST 5.5.1
    http://www6.software.ibm.com/reg/vastk/vastk551-i
- General
  - Smalltalk Language Newsgroup
    comp.lang.smalltalk
  - Instantiations' Smalltalk Web Site
    http://www.instantiations.com/sts
  - Me ;-)
    mailto:clayberg@instantiations.com