Class: **AdventureActorBiographyViewer**

Superclass: AdventureApplicationModel
Category: Adventure-Views
Instance variables: actor

I display my #actor's bio. When its adventure ends, I close.

Instance Variables:
actor <AdventureActor> who's bio I'm displaying.

accessing

actor
```
^actor
```

actor: anActor
```
actor := anActor
```

aspects

biographyText
```
^self actor biography asText asValue
```

interface opening

postBuildWith: aBuilder
```
(aBuilder componentAt: #actorNameLabel)
    labelString: self actor name.
(aBuilder componentAt: #actorImageLabel)
    label: self actor image
```

initialize-release

on: anActor
```
self actor: anActor.
self restoreDependency.
self registerDependency
```

view creation

openOn: anActor
```
self on: anActor.
self open
```

dependency control

adventure
```
^self actor adventure
```

view updating

changeRequest
^self simpleChangeRequest

MetaClass: AdventureActorBiographyViewer class

interface specs

windowSpec
"UIPainter new openOnClass: self andSelector: #windowSpec"

^#(#FullSpec #window: #(#WindowSpec #label: 'Actor Biography Viewer' #min: #(#Point 374 186) #bounds: #(#Rectangle 385 340 794 645)) #component: #(#SpecCollection #collection: #(#(#TextEditorSpec #layout: #(#LayoutFrame 8 0 88 0 -8 1 -8 1) #model: #biographyText #isReadOnly: true) #(#LabelSpec #layout: #(#Point 8 8) #name: #actorImageLabel #label: '(image)') #(#LabelSpec #layout: #(#Point 88 8) #name: #actorNameLabel #label: '(name)' #style: #large))))

view creation

openOn: anActor
^self new openOn: anActor

Class: **AdventureApplicationModel**

Superclass: ApplicationModel
Category: Adventure-Views

I add default dependency control and #changeRequest handlers to ApplicationModel. I set up the application to close when the adventure #isOver.

Subclasses must implement methods for
accessing
adventure

dependency control

breakDependency
```
self adventure retractInterestIn: #isOver for: self
```

deregisterDependency
```
self adventure dependencyCoordinator
    deregister: self
    onChannel: #default
    asExternalDependentOf: self adventure
```

registerDependency
```
self adventure dependencyCoordinator
    register: self
    onChannel: #default
    asExternalDependentOf: self adventure
    breakSelector: #breakDependency
    restoreSelector: #restoreDependency
```

restoreDependency
```
self adventure
    expressInterestIn: #isOver
    for: self
    sendBack: #closeRequest
```

view updating

simpleChangeRequest
```
"Either the user is attempting to exit via 'close' on the window menu,
or the game is over."

^super changeRequest
    and:
        [self breakDependency.
        self deregisterDependency.
        true]
```

accessing

adventure

```
self subclassResponsibility
```

MetaClass: AdventureApplicationModel class

Class: **AdventureContainerInspector**

Superclass: ApplicationModel
Category: Adventure-Views
Instance variables: container player contentsList lastSelection selectedItemDescription selectionDependency spinDelay
Class variables: NilSelectionImage

I inspect any sort of AdventureSimpleContainer (or subclass instances.) I'm made to be embedded in a UI via a view holder. Therefore, I expect my client to send me #preClose just before it closes, so I can break my dependencies. I am wired to register with the adventure's dependency coordinator.

Instance Variables:
container <AdventureSimpleContainer> that I am inspecting.
player <AdventurePlayer> for whom I am inspecting.
contentsList <SelectionInList> UI aspect for my container's contents. Its #list is a List holding AdventureIndentedThingHolder's.
lastSelection <AdventureThing | nil> that was last selected by the user. This is maintained so that I can re-select as appropriate after list updates.
selectedItemDescription <ValueHolder> on a String. The UI aspect for the description of the selected item.
selectionDependency <AdventureThing | nil> A record of what, if anything, I am dependent on as the list selection. Holding this allows me to easily break and restore that dependency.
spinDelay <Delay> used when I have to spin waiting for a turn.

Class Variables:
NilSelectionImage <CachedImage> that is displayed when nothing is selected.

aspects

contentsList
```
^contentsList isNil
    ifTrue: [contentsList := SelectionInList new]
    ifFalse: [contentsList]
```

selectedItemDescription
```
^selectedItemDescription isNil
    ifTrue: [selectedItemDescription := String new asValue]
    ifFalse: [selectedItemDescription]
```

printing

printOn: aStream
```
"Append a representation of myself to aStream."

super printOn: aStream.
aStream nextPutAll: ' on: ' , (self container isNil ifFalse: [self container name]
            ifTrue: [nil]) printString
```

initialize-release

breakContainerDependency

```
""
"Transcript show: 'breakContainerDependency by ' , self printString; cr."

self container retractInterestIn: #contents for: self.
self container retractInterestIn: #name for: self
```

deregisterContainerDependency

```
self player adventure dependencyCoordinator
    deregister: self
    onChannel: #container
    asExternalDependentOf: self player adventure
```

disconnectFromContainer

```
self breakContainerDependency.
self deregisterContainerDependency
```

on: aContainer

```
"Change my container. My #player must already be set. This should only
be sent after the window is open, since it generates a
display update."

self container notNil
    ifTrue:
        [self breakContainerDependency.
        self deregisterContainerDependency].
container := aContainer.
self restoreContainerDependency.
self registerContainerDependency.
self updateContentsList
```

on: aContainer player: aPlayer

```
"This should only be sent after the window is open, since it generates a
display update."

self player: aPlayer.
self on: aContainer
```

preClose

```
"This is sent just before our client closes."

self ensureNoSelectionDependency.
self disconnectFromContainer
```

registerContainerDependency

```
self player adventure dependencyCoordinator
    register: self
    onChannel: #container
    asExternalDependentOf: self player adventure
    breakSelector: #breakContainerDependency
    restoreSelector: #restoreContainerDependency
```

restoreContainerDependency

```
""
"Transcript show: 'restoreContainerDependency by ' , self printString; cr."
```

```
self container
    expressInterestIn: #contents
    for: self
    sendBack: #updateContentsList.
self container
    expressInterestIn: #name
    for: self
    sendBack: #updateContentsList
```

interface opening

postBuildWith: aBuilder

```
"Set up the dependency on the list selection holder here because 'self
builder' must be non-nil for #listSelectionChanged to update the
selected object image."

self contentsList selectionIndexHolder onChangeSend: #listSelectionChanged to: self
```

list updating

add: aThing to: aCollection atIndentLevel: anInteger

```
aCollection add: (AdventureIndentedThingHolder indentLevel: anInteger thing: aThing).
(aThing isContainer and: [aThing contentsExposed])
    ifTrue: [aThing contents do: [:thing | self
                add: thing
                to: aCollection
                atIndentLevel: anInteger + 1]]
```

setContentsList

```
"Set the new contents list."

| newList |
newList := List new.
self
    add: self container
    to: newList
    atIndentLevel: 0.
self contentsList list: newList
```

updateContentsList

```
"My container (which is included in my display) or its contents have
changed, and I must redisplay them. If the current selection is still in the
list, I restore it after the redisplay."

self setContentsList.
self restoreSelectionIfPossible
```

menus

addMenuItemsIn: mCollection ofType: aSymbol toLabels: lCollection values: vCollection

```
| items |
items := mCollection select: [:item | item type == aSymbol].
items
    do:
```

```
            [:item |
            lCollection add: item label.
            vCollection
                add:
                    [item block value.
                    self player adventure endTurn]]
```

contentsListMenu

```
    ^AdventureDynamicPopUpMenu startUpBlock: self startUpBlock noItemSelectedBlock: self
noItemSelectedBlock
```

noItemSelectedBlock

```
    ^[:aMenu | self player adventure endTurn]
```

startUpBlock

```
    ^
    [:aMenu |
    | labels lines values menuItems normalItemsSize gameIsOver |
    labels := OrderedCollection new.
    lines := OrderedCollection new.
    values := OrderedCollection new.
    self hasSelection
        ifTrue:
            [gameIsOver := self player adventure beginTurnAndTestGameOverWithSpinDelay: self
spinDelay.
            (gameIsOver | self player gameIsOver) not
                ifTrue:
                    [menuItems := self lastSelection menuItemsFor: self player.
                    self
                        addMenuItemsIn: menuItems
                        ofType: #normal
                        toLabels: labels
                        values: values.
                    normalItemsSize := labels size.
                    self lastSelection debugMenusEnabled
                        ifTrue:
                            [self
                                addMenuItemsIn: menuItems
                                ofType: #debug
                                toLabels: labels
                                values: values.
                            normalItemsSize > 0 ifTrue: [lines add: normalItemsSize].
                            labels add: 'inspect interface'.
                            values add: [self inspect]]]].
    aMenu
        labelArray: labels
        lines: lines
        values: values]
```

accessing

container

```
    ^container
```

lastSelection

```
    ^lastSelection
```

lastSelection: aThing
```
lastSelection := aThing
```

player
```
^player
```

player: aPlayer
```
player := aPlayer
```

selectionDependency
```
^selectionDependency
```

selectionDependency: anObject
```
selectionDependency := anObject
```

spinDelay
```
^spinDelay isNil
    ifTrue: [spinDelay := Delay forMilliseconds: 80]
    ifFalse: [spinDelay]
```

spinDelay: aDelay
```
spinDelay := aDelay
```

selection updating

listSelectionChanged
```
self updateSelection.
self updateSelectedItemImageLabel.
self updateSelectedItemDescription
```

updateSelectedItemDescription
```
self selectedItemDescription value: (self hasSelection
        ifTrue: [self contentsList selection thing description]
        ifFalse: [String new])
```

updateSelectedItemImageLabel
```
"Display the image for the current selection. Do nothing if the new image
is the same as the old image to avoid display flashing."

| imageComponent oldImage newImage |
imageComponent := self builder componentAt: #selectedItemImageLabel.
oldImage := imageComponent widget label.
newImage := self hasSelection
            ifTrue: [self contentsList selection thing image]
            ifFalse: [self class nilSelectionImage].
oldImage ~= newImage ifTrue: [imageComponent label: newImage]
```

updateSelection
```
"(self container isKindOf: AdventurePlace)
    ifTrue: [Transcript show: 'has selection ' , self hasSelection printString; cr; show: 'last selection: ' ,
self lastSelection printString; cr; cr]."
self hasSelection
    ifTrue:
        [self ensureSelectionDependency.
        self setLastSelection]
```

```
		ifFalse: [self ensureNoSelectionDependency]
```

remember last selection

restoreSelectionIfPossible

```
	"The name here says it all. When a thing is moved from one container to
	another, there is a state where the thing is not in any container--at that
	point, the selection goes nil, which is why we play this lastSelection
	game."

	self contentsList list do: [:holder | holder thing == self lastSelection ifTrue: [^self contentsList
selection: holder]].
	self contentsList selectionIndexHolder value: 1
```

setLastSelection

```
	self lastSelection: self currentSelection
```

selection dependency

breakAnySelectionDependency

```
	self selectionDependency notNil ifTrue: [self breakSelectionDependency]
```

breakSelectionDependency

```
	""
	"Transcript show: 'breakSelectionDependency by ' , self printString; cr."

	self selectionDependency retractInterestIn: #description for: self
```

currentSelection

```
	"Answer the currently selected thing, or nil."

	^self hasSelection
		ifTrue: [self contentsList selection thing]
		ifFalse: [nil]
```

deregisterSelectionDependency

```
	self player adventure dependencyCoordinator
		deregister: self
		onChannel: #selection
		asExternalDependentOf: self player adventure
```

ensureNoSelectionDependency

```
	"Ensure that I am not dependent on any selection."

	self selectionDependency notNil
		ifTrue:
			[self breakSelectionDependency.
			self deregisterSelectionDependency.
			self selectionDependency: nil]
```

ensureSelectionDependency

```
	"Ensure that I am dependent on my currentSelection."

	self currentSelection ~~ self selectionDependency
		ifTrue:
```

```
		[self ensureNoSelectionDependency.
		self selectionDependency: self currentSelection.
		self restoreSelectionDependency.
		self registerSelectionDependency]
```

hasSelection

```
	^self contentsList selection notNil
```

registerSelectionDependency

```
	self player adventure dependencyCoordinator
		register: self
		onChannel: #selection
		asExternalDependentOf: self player adventure
		breakSelector: #breakAnySelectionDependency
		restoreSelector: #restoreAnySelectionDependency
```

restoreAnySelectionDependency

```
	self selectionDependency notNil ifTrue: [self restoreSelectionDependency]
```

restoreSelectionDependency

```
	""
	"Transcript show: 'restoreSelectionDependency by ' , self printString; cr."

	self selectionDependency
		expressInterestIn: #description
		for: self
		sendBack: #updateSelectedItemDescription
```

MetaClass: AdventureContainerInspector class

interface specs

windowSpec

```
	"UIPainter new openOnClass: self andSelector: #windowSpec"

	^#(#FullSpec #window: #(#WindowSpec #label: 'Container Inspector' #min: #(#Point 288 192 )
#bounds: #(#Rectangle 480 498 768 690 ) ) #component: #(#SpecCollection #collection:
#(#(#SequenceViewSpec #layout: #(#LayoutFrame 0 0 0 0 0 1 0 0.541667 ) #model: #contentsList
#menu: #contentsListMenu ) #(#TextEditorSpec #layout: #(#LayoutFrame 88 0 16 0.541667 0 1 0 1 )
#model: #selectedItemDescription #isReadOnly: true ) #(#CompositeSpecCollection #collection:
#(#(#GroupBoxSpec #layout: #(#LayoutFrame 0 0 0 0 72 0 0 1.0 ) ) #(#LabelSpec #layout:
#(#LayoutFrame 2 0 0 0.0277778 67 0 0 0.930556 ) #name: #selectedItemImageLabel #label:
'nilSelectionImage' #hasCharacterOrientedLabel: false ) ) #compositeSpec: #(#CompositeSpec #layout:
#(#LayoutFrame 0 0 16 0.541667 72 0 88 0.541667 ) ) ) ) ) )
```

initialization

flushImageCache

```
	"self flushImageCache"

	NilSelectionImage := nil
```

resources

nilSelectionImage

```
^NilSelectionImage isNil
    ifTrue: [NilSelectionImage := CachedImage on: (Adventure readImageFromBOSSFile: 'nil.bos')]
    ifFalse: [NilSelectionImage]
```

Class: **AdventureControlPanel**

Superclass: AdventureApplicationModel
Category: Adventure-Views
Instance variables: adventure player playerInspector placeInspector playerMonitor lastSaveFileName spinDelay isOpen ledTimestamp
Class variables: LEDOffImage LEDOnImage

I am the interface to a running adventure.

Instance Variables:
adventure <Adventure> that I’m manipulating.
player <AdventurePlayer> that represents the physical user using me.
playerInspector <AdventureContainerInspector> UI subcanvas aspect for the inspector on my player.
placeInspector <AdventurePlaceInspector> UI subcanvas aspect for the inspector on my player’s enclosing container.
playerMonitor <AdventurePlayerMonitor> UI subcanvas aspect for the monitor on my player.
lastSaveFileName <String | nil> Holds the last file name I saved under, or nil if I haven’t saved.
spinDelay <Delay> used to spin when I must wait for a turn.
isOpen <Boolean> indicating that I am open, and therefore, I can be closed.
ledTimestamp <Integer> timestamp in milliseconds of the time at which my LED changed state. I use this to guarantee that a LED flash is visible.

Class Variables:
LEDOnImage <CachedImage> for a LED in the on state. I use LEDs to indicate turn activity.
LEDOffImage <CachedImage> for a LED in the off state.

actions

pauseGame
```
"Pause the game. If it’s already paused, do nothing."

self adventure pauseAndTerminateAllProcesses
```

resumeGame
```
self adventure resumeAllProcesses
```

interface opening

postBuildWith: aBuilder
```
aBuilder window label: self adventure name , ' - ' , self player name
```

postOpenWith: aBuilder
```
self placeInspector on: self player container player: self player.
self playerInspector on: self player player: self player.
self adventure postOpenWith: self.
AdventureCopyrightDialog openOn: self adventure product.
self resumeGame.
self isOpen: true
```

accessing

adventure

```
^adventure
```

adventure: anAdventure

```
adventure := anAdventure
```

isOpen

```
^isOpen isNil
    ifTrue: [isOpen := false]
    ifFalse: [isOpen]
```

isOpen: aBoolean

```
isOpen := aBoolean
```

lastSaveFileName

```
^lastSaveFileName
```

lastSaveFileName: aString

```
lastSaveFileName := aString
```

ledTimestamp

```
^ledTimestamp isNil
    ifTrue: [ledTimestamp := 0]
    ifFalse: [ledTimestamp]
```

ledTimestamp: anInteger

```
ledTimestamp := anInteger
```

player

```
^player
```

player: aPlayer

```
player := aPlayer
```

spinDelay

```
^spinDelay isNil
    ifTrue: [spinDelay := Delay forMilliseconds: 80]
    ifFalse: [spinDelay]
```

spinDelay: aDelay

```
spinDelay := aDelay
```

aspects

placeInspector

```
^placeInspector isNil
    ifTrue: [placeInspector := AdventurePlaceInspector new]
    ifFalse: [placeInspector]
```

playerInspector

```
^playerInspector isNil
    ifTrue: [playerInspector := AdventureContainerInspector new]
    ifFalse: [playerInspector]
```

playerMonitor

```
^playerMonitor isNil
    ifTrue: [playerMonitor := AdventurePlayerMonitor new]
    ifFalse: [playerMonitor]
```

initialize-release

on: anAdventure player: aPlayer

```
"Initialize myself on anAdventure, using aPlayer."

self adventure: anAdventure.
self player: aPlayer.
self player adventure: self adventure.
self playerMonitor on: self player.
self restoreDependency.
self registerDependency
```

player outcomes

displayPlayerDeathDialog

```
DialogView warn: 'You are dead. The game is over.'
```

displayPlayerLossDialog

```
DialogView warn: 'Dang! You have lost. The game is over.'
```

displayPlayerWinDialog

```
DialogView warn: 'Yippee. You have won. The game is over.'
```

playerHasLost

```
self displayPlayerLossDialog.
self closeRequest
```

playerHasWon

```
self displayPlayerWinDialog.
self closeRequest
```

playerIsDead

```
self displayPlayerDeathDialog.
self closeRequest
```

view updating

changeRequest

```
"Either the user is attempting to exit via 'close' on the window menu, the
entire game is over (#isOver), or the game is over for the player (win,
loss, or death). If the game is over, don't bother to ask the player if it
wants to exit. I use a new spin delay here, instead of my own, because a
race condition exists between a robot ending the game and the player
using me."

| gameIsOver |
"true ifTrue:[^true]."
gameIsOver := self adventure beginTurnAndTestGameOverWithSpinDelay: (Delay forMilliseconds:
250).
self isOpen
```

```
			ifFalse:
				["If we're already closed or not yet open, do nothing."
				self adventure endTurn.
				^false].
	gameIsOver ifTrue: [super changeRequest
				ifTrue:
					[self cleanUp.
					self adventure endTurn.
					^true]
				ifFalse: [^self error: 'unexpected condition']].
	self player gameIsOver ifTrue: [super changeRequest
				ifTrue:
					[self cleanUp.
					self adventure endGame.
					self adventure endTurn.
					^true]
				ifFalse: [^self error: 'unexpected condition']].
	(Dialog confirm: 'Are you sure you want to\exit ' withCRs , self adventure name , '?' initialAnswer: false)
		ifTrue:
			[self cleanUp.
			self adventure endGame.
			self adventure endTurn.
			^true]
		ifFalse:
			[self adventure endTurn.
			^false]
```

cleanUp

```
	self breakDependency.
	self deregisterDependency.
	self placeInspector preClose.
	self playerInspector preClose.
	self playerMonitor preClose.
	self isOpen: false
```

turnInProgress: aBoolean

```
	| additionalDelay |
	additionalDelay := self ledTimestamp + 50 - Time millisecondClockValue.
	additionalDelay > 0 ifTrue: [(Delay forMilliseconds: additionalDelay) wait].
	self ledTimestamp: Time millisecondClockValue.
	(self builder componentAt: #activityLED)
		label: (aBoolean
				ifTrue: [self class ledOnImage]
				ifFalse: [self class ledOffImage])
```

updateContainerInspector

```
	self placeInspector on: self player container
```

menus

debugMenu

```
	^AdventureDynamicPopUpMenu startUpBlock: self debugMenuStartUpBlock noItemSelectedBlock: self noItemSelectedBlock
```

debugMenuStartUpBlock

```
    "Transcript cr; show: 'Debug menu startUpBlock: ' , aMenu printString"

    ^
    [:aMenu |
    | labels lines values gameIsOver |
    labels := OrderedCollection new.
    lines := OrderedCollection new.
    values := OrderedCollection new.
    gameIsOver := self adventure beginTurnAndTestGameOverWithSpinDelay: self spinDelay.
    (gameIsOver | self player gameIsOver) not
        ifTrue:
            [self adventure debugMenusEnabled
                ifTrue:
                    [labels add: 'disable debug menus'.
                    values
                        add:
                            [self adventure debugMenusEnabled: false.
                            self adventure endTurn]]
                ifFalse:
                    [labels add: 'enable debug menus'.
                    values
                        add:
                            [self adventure debugMenusEnabled: true.
                            self adventure endTurn]].
            labels add: 'inspect interface'.
            values
                add:
                    [self inspect.
                    self adventure endTurn].
            labels add: 'spawn adventure'.
            values
                add:
                    [AdventureStandaloneContainerInspector openOn: self adventure player: self player.
                    self adventure endTurn]].
    aMenu
        labelArray: labels
        lines: lines
        values: values]
```

gameMenu

```
    ^AdventureDynamicPopUpMenu startUpBlock: self gameMenuStartUpBlock noItemSelectedBlock:
self noItemSelectedBlock
```

gameMenuStartUpBlock

```
    ^
    [:aMenu |
    | labels lines values gameIsOver |
    labels := OrderedCollection new.
    lines := OrderedCollection new.
    values := OrderedCollection new.
    gameIsOver := self adventure beginTurnAndTestGameOverWithSpinDelay: self spinDelay.
    (gameIsOver | self player gameIsOver) not
        ifTrue:
            [self adventure isPaused
                ifTrue:
                    [labels add: 'resume'.
```

```
                values
                    add:
                        [self resumeGame.
                        self adventure endTurn]]
            ifFalse:
                [labels add: 'pause'.
                values
                    add:
                        [self pauseGame.
                        self adventure endTurn]].
        labels add: 'save as...'.
        values add: [self executeSaveOperation: [self saveAs]].
        self lastSaveFileName notNil
            ifTrue:
                [labels add: 'save as ' , self lastSaveFileName.
                values add: [self executeSaveOperation: [self save]]]].
    aMenu
        labelArray: labels
        lines: lines
        values: values]
```

helpMenu

```
    ^AdventureDynamicPopUpMenu startUpBlock: self helpMenuStartUpBlock noItemSelectedBlock: self
noItemSelectedBlock
```

helpMenuStartUpBlock

```
    ^
    [:aMenu |
    | labels lines values gameIsOver |
    labels := OrderedCollection new.
    lines := OrderedCollection new.
    values := OrderedCollection new.
    gameIsOver := self adventure beginTurnAndTestGameOverWithSpinDelay: self spinDelay.
    (gameIsOver | self player gameIsOver) not
        ifTrue:
            [labels add: 'instructions...'.
            values
                add:
                    [AdventureHelpViewer openOn: self adventure product viewing: #instructions.
                    self adventure endTurn].
            labels add: 'about...'.
            values
                add:
                    [AdventureHelpViewer openOn: self adventure product viewing: #about.
                    self adventure endTurn]].
    aMenu
        labelArray: labels
        lines: lines
        values: values]
```

noItemSelectedBlock

```
    ^[:aMenu | self adventure endTurn]
```

view creation

openOn: anAdventure player: aPlayer

```
	"Open myself on anAdventure, using aPlayer."

	self on: anAdventure player: aPlayer.
	self open
```

dependency control

breakDependency

```
	self adventure retractInterestIn: #isOver for: self.
	self adventure retractInterestIn: #turnInProgress for: self.
	self player retractInterestIn: #hasDied for: self.
	self player retractInterestIn: #hasWon for: self.
	self player retractInterestIn: #hasLost for: self.
	self player retractInterestIn: #container for: self
```

restoreDependency

```
	self adventure
		expressInterestIn: #isOver
		for: self
		sendBack: #closeRequest.
	self adventure
		expressInterestIn: #turnInProgress
		for: self
		sendBack: #turnInProgress:.
	self player
		expressInterestIn: #hasDied
		for: self
		sendBack: #playerIsDead.
	self player
		expressInterestIn: #hasWon
		for: self
		sendBack: #playerHasWon.
	self player
		expressInterestIn: #hasLost
		for: self
		sendBack: #playerHasLost.
	self player
		expressInterestIn: #container
		for: self
		sendBack: #updateContainerInspector
```

saving

executeSaveOperation: aBlock

```
	"Pause to end turn competition and to sever references to processes
	(which would prevent a BOSS.) End the turn before executing the
	operation so the adventure's reference to us is broken, thus
	allowing the BOSS to work."

	self adventure
		executePaused:
			[self adventure endTurn.
			aBlock value]
```

save

```
    "Assume that the adventure is paused and that there are no turn requests
    pending."

    self saveAs: self lastSaveFileName
```

saveAs

```
    "Assume that the adventure is paused and that there are no turn requests
    pending."

    | file |
    file := self saveAsFileName.
    file isEmpty ifFalse: [self saveAs: file]
```

saveAs: aString

```
    "Assume that the adventure is paused and that there are no turn requests
    pending."

    | stream bos file actualFile |
    file := aString.
    Object errorSignal
        handle:
            [:ex |
            bos notNil
                ifTrue: [bos close]
                ifFalse: [stream notNil ifTrue: [stream close]].
            Dialog warn: 'Can''t save to ' , file , ':\' withCRs , ex errorString.
            file := self saveAsFileName.
            file isEmpty ifFalse: [ex restart]]
        do: [Cursor write
                showWhile:
                    [stream := nil.
                    bos := nil.
                    stream := file asFilename writeStream.
                    actualFile := stream name.
                    bos := BinaryObjectStorage onNew: stream.
                    self adventure save: self player on: bos.
                    bos close.
                    self lastSaveFileName: actualFile]]
```

saveAsFileName

```
    "Believe it or not, #requestFileName:default:version: blows up if you give
    it 'lpt1:' (or other device name) under OS/2"

    | file suggestion |
    suggestion := self adventure suggestSaveFile.
    file := Object errorSignal
                handle:
                    [:ex |
                    Dialog warn: 'Can''t use the entered filename:\' withCRs , ex errorString.
                    ex restart]
                do: [AdventureWideDialog
                        requestFileName: 'Enter name of save file:'
                        default: suggestion
                        version: #new].
    file = suggestion ifTrue: [self adventure updateSaveFileSuggestion].
    ^file
```

MetaClass: AdventureControlPanel class

interface specs

windowSpec

```
	"UIPainter new openOnClass: self andSelector: #windowSpec"

	^#(#FullSpec #window: #(#WindowSpec #label: 'Adventure Control Panel' #min: #(#Point 640 480 ) #bounds: #(#Rectangle 283 246 923 726 ) #flags: 8 ) #component: #(#SpecCollection #collection: #(#(#SubCanvasSpec #layout: #(#LayoutFrame 8 0 48 0 0 0.485849 -8 1 ) #flags: 0 #majorKey: #AdventurePlaceInspector #minorKey: #windowSpec #clientKey: #placeInspector ) #(#SubCanvasSpec #layout: #(#LayoutFrame 16 0.487644 8 0 -8 1 0 0.512605 ) #flags: 0 #majorKey: #AdventureContainerInspector #minorKey: #windowSpec #clientKey: #playerInspector ) #(#SubCanvasSpec #layout: #(#LayoutFrame 16 0.486043 0 0.546218 -8 1 0 1 ) #flags: 0 #majorKey: #AdventurePlayerMonitor #minorKey: #windowSpec #clientKey: #playerMonitor ) #(#CompositeSpecCollection #collection: #(#(#MenuButtonSpec #layout: #(#LayoutFrame 0 0 0 0 80 0 24 0 ) #label: 'Game' #menu: #gameMenu #isPopUp: true ) #(#MenuButtonSpec #layout: #(#LayoutFrame 176 0 0 0 256 0 24 0 ) #label: 'Special' #menu: #debugMenu #isPopUp: true ) #(#MenuButtonSpec #layout: #(#LayoutFrame 88 0 0 0 168 0 24 0 ) #label: 'Help' #menu: #helpMenu #isPopUp: true ) ) #compositeSpec: #(#CompositeSpec #layout: #(#Rectangle 53 8 309 32 ) ) ) #(#LabelSpec #layout: #(#Point 8 8 ) #name: #activityLED #label: #ledOffImage #hasCharacterOrientedLabel: false ) ) ) )
```

view creation

openOn: anAdventure player: aPlayer

```
	"Open an instance of myself on anAdventure, using aPlayer."

	^self new openOn: anAdventure player: aPlayer
```

resources

flushImageCache

```
	""
	"self flushImageCache"

	LEDOnImage := nil.
	LEDOffImage := nil
```

ledOffImage

```
	^LEDOffImage isNil
		ifTrue: [LEDOffImage := CachedImage on: (Adventure readImageFromBOSSFile: 'led_off.bos')]
		ifFalse: [LEDOffImage]
```

ledOnImage

```
	^LEDOnImage isNil
		ifTrue: [LEDOnImage := CachedImage on: (Adventure readImageFromBOSSFile: 'led_on.bos')]
		ifFalse: [LEDOnImage]
```

Class: **AdventureCopyrightDialog**

Superclass: AdventureProductInformationDialog
Category: Adventure-Views

I’m the copyright dialog for an AdventureProduct.

MetaClass: AdventureCopyrightDialog class

interface specs

windowSpec

"UIPainter new openOnClass: self andSelector: #windowSpec"

^#(#FullSpec #window: #(#WindowSpec #label: ’Copyright Dialog’ #min: #(#Point 351 336) #max: #(#Point 351 336) #bounds: #(#Rectangle 336 249 687 585) #colors: #(#LookPreferences #setForegroundColor: #(#ColorValue #black) #setBackgroundColor: #(#ColorValue 6553 6553 6553) #setSelectionForegroundColor: #(#ColorValue #black) #setSelectionBackgroundColor: #(#ColorValue 4915 4915 4915) #setBorderColor: #(#ColorValue #black))) #component: #(#SpecCollection #collection: #(#(#TextEditorSpec #layout: #(#Rectangle 8 104 344 280) #flags: 4 #model: #creditsText #alignment: #center #isReadOnly: true) #(#LabelSpec #layout: #(#Point 8 8) #name: #productLogoLabel #label: ’(logo)’) #(#TextEditorSpec #layout: #(#Rectangle 80 32 344 104) #flags: 4 #model: #copyrightText #style: #small #isReadOnly: true) #(#TextEditorSpec #layout: #(#Rectangle 80 8 344 32) #flags: 4 #model: #productNameAndVersionText #style: #large #isReadOnly: true) #(#ActionButtonSpec #layout: #(#Rectangle 132 279 220 327) #model: #accept #label: ’Ok’ #defaultable: true))))

Class: **AdventureCredit**

Superclass: Object
Category: Adventure-Views
Instance variables: role name additionalInformation

I hold information about a contributor who deserves credit.

Instance Variables:
role <String> The contributor’s role, such as "Design and Programming".
name <String> The contributor’s name.
additionalInformation <String> Optional. Place an email address or some other information here.

accessing

additionalInformation
```
^additionalInformation isNil
    ifTrue: [additionalInformation := String new]
    ifFalse: [additionalInformation]
```

additionalInformation: aString
```
additionalInformation := aString
```

name
```
^name
```

name: aString
```
name := aString
```

role
```
^role
```

role: aString
```
role := aString
```

MetaClass: AdventureCredit class

instance creation

role: roleString name: nameString additionalInformation: aiString
```
^(self new) role: roleString; name: nameString; additionalInformation: aiString
```

Class: **AdventureDetonator**

Superclass: AdventureApplicationModel
Category: Adventure-Views
Instance variables: adventure

I'm a debugging aid. Under certain circumstances, it's very convenient to waste the game directly. Press my button, and I tell the adventure to toast itself without regard to the usual synchronization (which is probably broken.)

Instance Variables:
adventure <Adventure> that I detonate.

actions

detonate
self adventure endGameWithoutSynchronization

accessing

adventure
^adventure

adventure: anAdventure
adventure := anAdventure

initialize-release

on: anAdventure
self adventure: anAdventure.
self restoreDependency.
self registerDependency

view creation

openOn: anAdventure
self on: anAdventure.
self open

view updating

changeRequest
^self simpleChangeRequest

MetaClass: AdventureDetonator class

interface specs

windowSpec

```
	"UIPainter new openOnClass: self andSelector: #windowSpec"

	^#(#FullSpec #window: #(#WindowSpec #label: 'Detonator' #min: #(#Point 128 48 ) #max: #(#Point 128 48 ) #bounds: #(#Rectangle 362 313 490 361 ) ) #component: #(#SpecCollection #collection: #(#(#ActionButtonSpec #layout: #(#Rectangle 8 8 120 40 ) #model: #detonate #label: 'Detonate' #defaultable: true ) ) ) )
```

view creation

openOn: anAdventure

```
	^self new openOn: anAdventure
```

Class: **AdventureDynamicPopUpMenu**

Superclass: PopUpMenu
Category: Adventure-Views
Instance variables: startUpBlock noItemSelectedBlock

I am a truly dynamic menu. I build myself after the user has done its mouse click to bring me up. I'm very converient when a menu's contents must depend on a complicated set of factors, or it is impractical to use a value holder for a menu.

Instance Variables:
startUpBlock <BlockClosure> Executed before the menu is built. This block is passed self (the PopUpMenu being built.) It should add all items to the menu with #labelArray:lines:values:.
noItemSelectedBlock <BlockClosure | nil> This block is optional. It is only executed if the user made no selection. It is passed self. his hook is useful if the startUpBlock is synching with something, and you must release the synch resources when the user decides not to select an item.

accessing

noItemSelectedBlock

```
^noItemSelectedBlock
```

noItemSelectedBlock: aBlock

```
noItemSelectedBlock := aBlock
```

startUpBlock

```
^startUpBlock
```

startUpBlock: aBlock

```
startUpBlock := aBlock
```

controlling

startUpAt: aPoint centerX: xBoolean centerY: yBoolean

```
"Display and make a selection from the receiver centering around the
marker according to xBoolean and yBoolean. Answer the current
selection."

| sel |
self startUpBlock value: self.
sel := super
		startUpAt: aPoint
		centerX: xBoolean
		centerY: yBoolean.
sel = 0 & self noItemSelectedBlock notNil ifTrue: [self noItemSelectedBlock value: self].
^sel
```

initialize-release

startUpBlock: sBlock noItemSelectedBlock: nBlock

```
self startUpBlock: sBlock.
self noItemSelectedBlock: nBlock
```

MetaClass: AdventureDynamicPopUpMenu class

instance creation

startUpBlock: sBlock noItemSelectedBlock: nBlock

```
^self new startUpBlock: sBlock noItemSelectedBlock: nBlock
```

Class: **AdventureHelpViewer**

Superclass: ApplicationModel
Category: Adventure-Views
Instance variables: product text textMode

I display information and instructions about an AdventureProduct. When you open me, you specify what product text to display initially; either #about, or #instructions.

Instance Variables:
product <AdventureProduct> who's help I'm viewing.
text <Text> currently selected by textMode.
textMode <ValueHolder> on a Symbol that designates the part of the product being viewed in my text area. textMode's value is sent as a selector to my product to fetch the text.

aspects

copyrightText
```
    ^self product copyright asText asValue
```

productNameAndVersionText
```
    ^(Text string: self product name , ' ' , self product version emphasis: self
productNameHighlightEmphasis) asValue
```

productNameHighlightEmphasis
```
    ^Array with: #bold with: #color -> ColorValue black
```

text
```
    ^text isNil
        ifTrue: [text := (self product perform: self textMode value) asValue]
        ifFalse: [text]
```

textMode
```
    ^textMode
```

textMode: aValueHolder
```
    textMode := aValueHolder
```

actions

setTextFromMode
```
    self text value: (self product perform: self textMode value)
```

initialize-release

on: aProduct viewing: aSymbol
```
    "aSymbol specifies what text in aProduct to view initially. It may be
    #about or #instructions."

    self product: aProduct.
    self textMode: aSymbol asValue.
```

```
	self textMode onChangeSend: #setTextFromMode to: self
```

view creation

openOn: aProduct viewing: aSymbol
```
	self on: aProduct viewing: aSymbol.
	self open
```

accessing

product
```
	^product
```

product: aProduct
```
	product := aProduct
```

interface opening

postBuildWith: aBuilder
```
	(aBuilder componentAt: #productLogoLabel)
		label: self product logo.
	aBuilder window label: self product name , ' Help'
```

MetaClass: AdventureHelpViewer class

interface specs

windowSpec
```
	"UIPainter new openOnClass: self andSelector: #windowSpec"

	^#(#FullSpec #window: #(#WindowSpec #label: 'Product Viewer' #min: #(#Point 447 208 ) #bounds: #(#Rectangle 200 338 672 687 ) ) #component: #(#SpecCollection #collection: #(#(#TextEditorSpec #layout: #(#LayoutFrame 8 0 104 0 -8 1 -8 1 ) #model: #text #isReadOnly: true ) #(#LabelSpec #layout: #(#Point 8 8 ) #name: #productLogoLabel #label: '(logo)' ) #(#TextEditorSpec #layout: #(#Rectangle 80 8 344 32 ) #flags: 4 #model: #productNameAndVersionText #style: #large #isReadOnly: true ) #(#TextEditorSpec #layout: #(#Rectangle 80 32 344 104 ) #flags: 4 #model: #copyrightText #style: #small #isReadOnly: true ) #(#MenuButtonSpec #layout: #(#LayoutFrame -96 1 8 0 -8 1 32 0 ) #model: #textMode #label: 'View' #menu: #textModeMenu #isPopUp: false ) ) ) )
```

view creation

openOn: aProduct viewing: aSymbol
```
	^self new openOn: aProduct viewing: aSymbol
```

resources

textModeMenu
```
	"UIMenuEditor new openOnClass: self andSelector: #textModeMenu"

	^#(#PopUpMenu #('about' 'instructions' ) #() #(#about #instructions ) ) decodeAsLiteralArray
```

Class: **AdventureIndentedThingHolder**

Superclass: Model
Category: Adventure-Views
Instance variables: thing indentLevel

I'm used to wrap AdventureThing's in some useful user-interface information. Actually, I just hold an indentLevel. I re-implement #displayString to indent my #thing's #displayString by indentLevel tabs.

Instance Variables:
thing <AdventureThing> that I'm wrapping.
indentLevel <Integer> number of my indent level. Level 0 is at the top of the hierarchy, and is not indented.

printing

displayString
```
| s |
s := (String new: 24) writeStream.
self indentLevel timesRepeat: [s tab].
s nextPutAll: self thing displayString.
^s contents
```

printOn: aStream
```
"Append a representation of myself to aStream."

super printOn: aStream.
aStream nextPutAll: ' on: ' , self thing printString
```

accessing

indentLevel
```
^indentLevel
```

indentLevel: anInteger
```
indentLevel := anInteger
```

thing
```
^thing
```

thing: aThing
```
thing := aThing
```

MetaClass: AdventureIndentedThingHolder class

instance creation

indentLevel: anInteger thing: aThing

```
^(self new) indentLevel: anInteger; thing: aThing
```

Class: **AdventureLauncher**

Superclass: ApplicationModel
Category: Adventure-Views

I provide the interface to start new Adventure's, and to open old Adventure's that have been saved to disk. I'm prepared to deal with all manner of I/O errors, and I bitch gracefully if the user tries to open an adventure with an obsolete binaryRepresentationVersion anywhere.

I find Adventure-building classes by searching all subclasses of Adventure. Anything that answers #true when sent #isStartable is added to my menus.

menus

fileMenu
```
^AdventureDynamicPopUpMenu startUpBlock: self fileMenuStartUpBlock noItemSelectedBlock: nil
```

fileMenuStartUpBlock
```
^
[:aMenu |
| labels values |
labels := OrderedCollection new.
values := OrderedCollection new.
labels add: 'new'.
values add: self newMenu.
labels add: 'open...'.
values add: [self openSavedGame].
aMenu
    labelArray: labels
    lines: Array new
    values: values]
```

helpMenu
```
^AdventureDynamicPopUpMenu startUpBlock: self helpMenuStartUpBlock noItemSelectedBlock: nil
```

helpMenuFor: aClass
```
| labels values |
labels := OrderedCollection new.
values := OrderedCollection new.
labels add: 'instructions'.
values add: [AdventureHelpViewer openOn: aClass product viewing: #instructions].
labels add: 'about'.
values add: [AdventureHelpViewer openOn: aClass product viewing: #about].
^PopUpMenu
    labelArray: labels
    lines: Array new
    values: values
```

helpMenuStartUpBlock
```
^
[:aMenu |
| labels values |
```

```
	labels := OrderedCollection new.
	values := OrderedCollection new.
	Adventure allSubclasses do: [:sub | sub isStartable
			ifTrue:
				[labels add: sub product name.
				values add: (self helpMenuFor: sub)]].
	aMenu
		labelArray: labels
		lines: Array new
		values: values]
```

newMenu

```
	| labels values |
	labels := OrderedCollection new.
	values := OrderedCollection new.
	Adventure allSubclasses do: [:sub | sub isStartable
			ifTrue:
				[labels add: sub product name.
				values add: [sub open]]].
	^PopUpMenu
		labelArray: labels
		lines: Array new
		values: values
```

interface opening

postOpenWith: aBuilder

```
	aBuilder window icon: (Icon constantNamed: #launcher)
```

view updating

changeRequest

```
	^super changeRequest and: [
		Dialog confirm: 'Are you sure you want to\close the launcher?' withCRs
			initialAnswer: false]
```

restoring

openSavedGame

```
	| file bos player stream |
	file := self savedFileName.
	file isEmpty ifTrue: [^self].
	Object errorSignal
		handle:
			[:ex |
			bos notNil
				ifTrue: [bos close]
				ifFalse: [stream notNil ifTrue: [stream close]].
			Dialog warn: 'Can''t read from ' , file , ':\' withCRs , ex errorString.
			file := self savedFileName.
			file isEmpty ifFalse: [ex restart]]
		do: [Cursor read
				showWhile:
					[stream := nil.
					bos := nil.
```

```
                stream := file asFilename readStream.
                bos := BinaryObjectStorage onOld: stream.
                player := bos next.
                bos close]].
    file isEmpty ifTrue: [^self].
    player adventure postRestoreAll.
    (player adventure class defaultUserInterface new) lastSaveFileName: file; openOn: player adventure
player: player.
    player adventure postRestorePostOpenAll
```

savedFileName

```
    "Believe it or not, #requestFileName:default:version: blows up if you give
    it 'lpt1:' (or other device name) under OS/2"

    ^Object errorSignal
        handle:
            [:ex |
            Dialog warn: 'Can''t use the entered filename:\' withCRs , ex errorString.
            ex restart]
        do: [Dialog
                requestFileName: 'Enter name of saved game:'
                default: '*' , Adventure suggestSaveFileSuffix
                version: #mustBeOld]
```

MetaClass: AdventureLauncher class

interface specs

windowSpec

```
    "UIPainter new openOnClass: self andSelector: #windowSpec"

    ^#(#FullSpec #window: #(#WindowSpec #label: 'Launcher - Adventures' #min: #(#Point 178 83 )
#max: #(#Point 178 83 ) #bounds: #(#Rectangle 246 317 424 400 ) ) #component: #(#SpecCollection
#collection: #(#(#LabelSpec #layout: #(#Point 8 8 ) #label: #icon #hasCharacterOrientedLabel: false )
#(#CompositeSpecCollection #collection: #(#(#MenuButtonSpec #layout: #(#LayoutFrame 0 0 0 0 80 0 24
0 ) #label: 'File' #menu: #fileMenu #isPopUp: true ) #(#MenuButtonSpec #layout: #(#LayoutFrame 0 0 32
0 80 0 56 0 ) #label: 'Help' #menu: #helpMenu #isPopUp: true ) ) #compositeSpec: #(#CompositeSpec
#layout: #(#Rectangle 88 8 168 64 ) ) ) ) ) ) )
```

resources

icon

```
    ^Adventure productLogo
```

Class: **AdventurePlaceInspector**

Superclass: AdventureContainerInspector
Category: Adventure-Views

*A PlaceInspector is identical to a ContainerInspector with one exception: no Player's contents are ever shown, because other Player's contents should be invisible and inaccessible. This feature anticipates multiple-players. *My* player is not shown at all, because it and its contents are already visible in the ContainerInspector on the player. (It was confusing to show the same player in two windows.)*

view updating

add: aThing to: aCollection atIndentLevel: anInteger

```
    "Completely supress the display of my player. Suppress the display of
    the contents of other players."

    aThing ~~ self player ifTrue: [(aThing isKindOf: AdventurePlayer)
            ifTrue: [aCollection add: (AdventureIndentedThingHolder indentLevel: anInteger thing:
aThing)]
            ifFalse: [super
                add: aThing
                to: aCollection
                atIndentLevel: anInteger]]
```

MetaClass: AdventurePlaceInspector class

Class: **AdventurePlayerMonitor**

Superclass: AdventureApplicationModel
Category: Adventure-Views
Instance variables: player earView positionAdaptor scoreAdaptor targetAdaptor

I display various aspects of a player's state, such as the contents of its #ear, its #position, etc. I'm designed to be embedded as a subcanvas. Therefore, I expect my client to send me #preClose just before it closes so I can break my dependencies.

Instance Variables:
player <AdventurePlayer> that I'm monitoring.
earView <TextCollectorView> that displays my player's #ear.
positionAdaptor <AspectAdaptor> for player aspect #position.
scoreAdaptor <AspectAdaptor> for player aspect #score.
targetAdaptor <AspectAdaptor> for player aspect #target.

initialize-release

on: aPlayer
```
self player: aPlayer.
self targetAdaptor: ((AspectAdaptor forAspect: #target)
        subject: self player).
self scoreAdaptor: ((AspectAdaptor forAspect: #score)
        subject: self player).
self positionAdaptor: ((AspectAdaptor forAspect: #position)
        subject: self player).
self earView: (TextCollectorView model: self player ear).
self restoreDependencyToAdaptors.
self registerDependency
```

preClose
```
"This is sent just before our client closes."

self breakDependency.
self deregisterDependency
```

accessing

earView
```
^earView
```

earView: aView
```
earView := aView
```

player
```
^player
```

player: aPlayer
```
player := aPlayer
```

positionAdaptor
    ^positionAdaptor

positionAdaptor: anAspectAdaptor
    positionAdaptor := anAspectAdaptor

scoreAdaptor
    ^scoreAdaptor

scoreAdaptor: anAspectAdaptor
    scoreAdaptor := anAspectAdaptor

targetAdaptor
    ^targetAdaptor

targetAdaptor: anAspectAdaptor
    targetAdaptor := anAspectAdaptor

view updating

earUpdated
```
(self builder componentAt: #earTopBorder) beVisible.
(self builder componentAt: #earBottomBorder) beVisible.
(Delay forMilliseconds: 150) wait.
(self builder componentAt: #earTopBorder) beInvisible.
(self builder componentAt: #earBottomBorder) beInvisible
```

dependency control

adventure
    ^self player adventure

breakDependency
```
self targetAdaptor subjectSendsUpdates: false.
self scoreAdaptor subjectSendsUpdates: false.
self positionAdaptor subjectSendsUpdates: false.
self player ear removeDependent: self earView.
self player ear retractInterestsFor: self
```

restoreDependency
```
self restoreDependencyToAdaptors.
self player ear addDependent: self earView
```

restoreDependencyToAdaptors
```
self targetAdaptor subjectSendsUpdates: true.
self scoreAdaptor subjectSendsUpdates: true.
self positionAdaptor subjectSendsUpdates: true.
self player ear onChangeSend: #earUpdated to: self
```

aspects

position
    ^BlockValue block: [:aPosition | aPosition x printString , ', ' , aPosition y printString , ', ' , aPosition z printString]

```
        arguments: (Array with: self positionAdaptor)
```

targetName

```
    ^BlockValue block: [:aThing | aThing notNil
            ifTrue: [aThing name]
            ifFalse: ['(select a target)']]
        arguments: (Array with: self targetAdaptor)
```

printing

printOn: aStream

```
    "Append a representation of myself to aStream."

    super printOn: aStream.
    aStream nextPutAll: ' on: ' , (self player isNil ifFalse: [self player name]
                ifTrue: [nil]) printString
```

interface opening

postBuildWith: aBuilder

```
    (aBuilder componentAt: #earTopBorder) beInvisible.
    (aBuilder componentAt: #earBottomBorder) beInvisible
```

MetaClass: AdventurePlayerMonitor class

interface specs

windowSpec

```
    "UIPainter new openOnClass: self andSelector: #windowSpec"

    ^#(#FullSpec #window: #(#WindowSpec #label: 'Player Monitor' #min: #(#Point 20 20 ) #bounds:
#(#Rectangle 364 365 652 573 ) ) #component: #(#SpecCollection #collection:
#(#(#ArbitraryComponentSpec #layout: #(#LayoutFrame 0 0 80 0 0 1.0 -8 1 ) #flags: 9 #component:
#earView ) #(#RegionSpec #layout: #(#LayoutFrame 0 0 -8 1 0 1 0 1 ) #name: #earBottomBorder
#isOpaque: true #colors: #(#LookPreferences #setBackgroundColor: #(#ColorValue #red ) ) )
#(#InputFieldSpec #layout: #(#LayoutFrame 224 0 0 0 288 0 24 0 ) #model: #scoreAdaptor #alignment:
#right #isReadOnly: true #type: #number ) #(#LabelSpec #layout: #(#LayoutOrigin 168 0 0 0 ) #label:
'Score:' ) #(#InputFieldSpec #layout: #(#LayoutFrame 72 0 0 0 152 0 24 0 ) #model: #position
#alignment: #right #isReadOnly: true #type: #string ) #(#LabelSpec #layout: #(#LayoutOrigin 0 0 0 0 )
#label: 'Position:' ) #(#InputFieldSpec #layout: #(#LayoutFrame 56 0 40 0 0 1 64 0 ) #model: #targetName
#isReadOnly: true ) #(#LabelSpec #layout: #(#LayoutOrigin 0 0 40 0 ) #label: 'Target:' ) #(#RegionSpec
#layout: #(#LayoutFrame 0 0 72 0 0 1 80 0 ) #name: #earTopBorder #isOpaque: true #colors:
#(#LookPreferences #setBackgroundColor: #(#ColorValue #red ) ) ) ) ) )
```

Class: **AdventureProductInformationDialog**

Superclass: SimpleDialog
Category: Adventure-Views
Instance variables: product

I’m an abstract superclass for dialogs that want to display information about an AdventureProduct.

Instance Variables:
product <AdventureProduct> that I display info about.

interface opening

postBuildWith: aBuilder
```
	(aBuilder componentAt: #productLogoLabel)
		label: self product logo
```

aspects

copyrightText
```
	^self product copyright asText asValue
```

creditsHighlightEmphasis
```
	^#()
```

creditsNormalEmphasis
```
	^#(#italic )
```

creditsText
```
	| text roleText nameText additionalText |
	text := Text new.
	self product credits
		do:
			[:credit |
			roleText := Text string: (credit role , ’\’) withCRs emphasis: self creditsNormalEmphasis.
			nameText := Text string: (credit name , ’\’) withCRs emphasis: self creditsHighlightEmphasis.
			additionalText := Text string: (credit additionalInformation , ’\\’) withCRs emphasis: self
creditsNormalEmphasis.
			text := text , roleText , nameText , additionalText].
	^text asValue
```

productNameAndVersionText
```
	^(Text string: self product name , ’ ’ , self product version emphasis: self
productNameHighlightEmphasis) asValue
```

productNameHighlightEmphasis
```
	^Array with: #bold with: #color -> ColorValue black
```

initialize-release

on: aProduct

```
    self product: aProduct
```

view creation

openOn: aProduct

```
    self on: aProduct.
    self open
```

accessing

product

```
    ^product
```

product: aProduct

```
    product := aProduct
```

MetaClass: AdventureProductInformationDialog class

view creation

openOn: aProduct

```
    ^self new openOn: aProduct
```

Class: **AdventureSayDialog**

Superclass: SimpleDialog
Category: Adventure-Views
Instance variables: sentenceField

I get a string from the user for my #player to say. If I answer #true, the value of my sentenceField value holder is what the user wants to say (self sentenceField value).

Instance Variables:
sentenceField <ValueHolder> on a String that holds what the user has input.

actions

acceptSentence
```
self accept value: true
```

say: aString
```
self sentenceField value: aString.
self acceptSentence
```

sayNo
```
self say: 'No.'
```

sayYes
```
self say: 'Yes.'
```

aspects

sentenceField
```
^sentenceField isNil
    ifTrue: [sentenceField := String new asValue]
    ifFalse: [sentenceField]
```

initialize-release

initialize
```
super initialize.
self sentenceField onChangeSend: #acceptSentence to: self
```

MetaClass: AdventureSayDialog class

interface specs

windowSpec
```
"UIPainter new openOnClass: self andSelector: #windowSpec"
```

```
^#(#FullSpec #window: #(#WindowSpec #label: 'Say Dialog' #min: #(#Point 20 20 ) #bounds: #(#Rectangle 406 370 802 515 ) ) #component: #(#SpecCollection #collection: #(#(#DividerSpec #layout: #(#Rectangle 8 72 384 76 ) ) #(#InputFieldSpec #layout: #(#Rectangle 8 8 288 32 ) #name: #sentenceField #model: #sentenceField #tabable: true ) #(#ActionButtonSpec #layout: #(#Rectangle 296 8 384 56 ) #model: #acceptSentence #label: 'Say' #defaultable: true ) #(#ActionButtonSpec #layout: #(#Rectangle 8 88 96 136 ) #model: #sayYes #label: 'Say Yes' #defaultable: true ) #(#ActionButtonSpec #layout: #(#Rectangle 112 88 200 136 ) #model: #sayNo #label: 'Say No' #defaultable: true ) #(#ActionButtonSpec #layout: #(#Rectangle 216 88 304 136 ) #model: #cancel #label: 'Cancel' #defaultable: true ) ) ) )
```

Class: **AdventureSingleStepper**

Superclass: AdventureApplicationModel
Category: Adventure-Views
Instance variables: robot

I’m the UI to single-step an AdventureRobot. I communicate with the robot via its "single stepping" protocol. I’m dependent on my robot’s adventure. When it ends, I close.

Instance Variables:
robot <AdventureRobot> that I’m stepping.

actions

proceed
```
self robot proceed.
self closeRequest
```

step
```
self buttons: #disable via: self builder.
self robot step
```

accessing

robot
```
^robot
```

robot: aRobot
```
robot := aRobot
```

interface opening

postBuildWith: aBuilder
```
aBuilder window label: self robot name.
self buttons: #disable via: aBuilder
```

view creation

openOn: aRobot
```
self on: aRobot.
self open
```

initialize-release

on: aRobot
```
self robot: aRobot.
self restoreDependency.
self registerDependency
```

view updating

buttons: aSelector via: aBuilder

```
(aBuilder componentAt: #stepButton)
    perform: aSelector.
(aBuilder componentAt: #proceedButton)
    perform: aSelector
```

changeRequest

```
"Either the user is attempting to exit via 'close' on the window menu,
the game is over, or the user has pressed the 'proceed' button."

^super changeRequest
    and:
        [self buttons: #disable via: self builder.
        self breakDependency.
        self deregisterDependency.
        true]
```

robotStepping

```
self buttons: #enable via: self builder
```

dependency control

adventure

```
^self robot adventure
```

breakDependency

```
super breakDependency.
self robot retractInterestIn: #singleStep for: self
```

restoreDependency

```
super restoreDependency.
self robot
    expressInterestIn: #singleStep
    for: self
    sendBack: #robotStepping
```

MetaClass: AdventureSingleStepper class

interface specs

windowSpec

```
"UIPainter new openOnClass: self andSelector: #windowSpec"

^#(#FullSpec #window: #(#WindowSpec #label: 'Single Stepper' #min: #(#Point 188 48 ) #max:
#(#Point 188 48 ) #bounds: #(#Rectangle 360 403 548 451 ) ) #component: #(#SpecCollection
#collection: #(#(#ActionButtonSpec #layout: #(#Rectangle 8 8 88 40 ) #name: #stepButton #model: #step
#label: 'Step' #defaultable: true ) #(#ActionButtonSpec #layout: #(#Rectangle 96 8 176 40 ) #name:
#proceedButton #model: #proceed #label: 'Proceed' #defaultable: true ) ) ) )
```

view creation

openOn: aRobot

```
^self new openOn: aRobot
```

Class: **AdventureStandaloneContainerInspector**

Superclass: AdventureApplicationModel
Category: Adventure-Views
Instance variables: container player containerInspector

I'm a freestanding AdventureContainerInspector, suitable for spawning from the adventure's UI. I'm dependent on the adventure, so I close when it ends.

Instance Variables:
container <AdventureSimpleContainer> that I'm inspecting.
player <AdventurePlayer> for whom I am inspecting.
containerInspector <AdventureContainerInspector> the inspector subcanvas.

interface opening

postBuildWith: aBuilder

```
"Hook into window opening at post-build time to set the name of the
window."

aBuilder window label: self container name
```

postOpenWith: aBuilder

```
self containerInspector on: self container player: self player
```

initialize-release

on: aContainer player: aPlayer

```
self container: aContainer.
self player: aPlayer.
self restoreDependency.
self registerDependency
```

aspects

containerInspector

```
^containerInspector isNil
    ifTrue: [containerInspector := AdventureContainerInspector new]
    ifFalse: [containerInspector]
```

view creation

openOn: aContainer player: aPlayer

```
self on: aContainer player: aPlayer.
self open
```

accessing

container

```
^container
```

container: aContainer

container := aContainer

player

^player

player: aPlayer

player := aPlayer

view updating

changeRequest

```
"Either the user is attempting to exit via 'close' on the window menu,
or the game is over"

^super changeRequest
    and:
        [self breakDependency.
        self deregisterDependency.
        self containerInspector preClose.
        true]
```

dependency control

adventure

^self player adventure

MetaClass: AdventureStandaloneContainerInspector class

interface specs

windowSpec

```
"UIPainter new openOnClass: self andSelector: #windowSpec"

^#(#FullSpec #window: #(#WindowSpec #label: 'Standalone Container Inspector' #min: #(#Point 20
20 ) #bounds: #(#Rectangle 330 301 635 508 ) ) #component: #(#SpecCollection #collection:
#(#(#SubCanvasSpec #layout: #(#LayoutFrame 0 0.0262295 0 0.0386473 0 0.970492 0 0.961353 )
#flags: 0 #majorKey: #AdventureContainerInspector #minorKey: #windowSpec #clientKey:
#containerInspector ) ) ) )
```

view creation

openOn: aContainer player: aPlayer

^self new openOn: aContainer player: aPlayer

Class: **AdventureWideDialog**

Superclass: Dialog
Category: Adventure-Views

I use an AdventureWideSimpleDialog instead of my superclass's use of SimpleDialog. This allows me to display dialogs (file dialogs particularly) that are wide enough that the user doesn't have to deal with PPS' crappy text-field editing.

MetaClass: AdventureWideDialog class

fill in the blank dialogs

request: messageString initialAnswer: aString onCancel: aBlockOrNil

```
"AdventureWideDialog request: 'Your name?' initialAnswer: 'name'
onCancel: ['Satan']"

^AdventureWideSimpleDialog new
    request: messageString
    initialAnswer: aString
    onCancel: aBlockOrNil
```

Class: **AdventureWideSimpleDialog**

Superclass: SimpleDialog
Category: Adventure-Views

I exist to serve AdventureWideDialog. I override my superclass’s #requestSpec to provide a much wider input area. This is so the user doesn’t have to deal with PPS’ crappy text-field editing.

MetaClass: AdventureWideSimpleDialog class

interface specs

requestSpec

```
"UIPainter new openOnClass: self andSelector: #requestSpec"

^#(#FullSpec #window: #(#WindowSpec #label: ' ' #min: #(#Point 396 110 ) #max: #(#Point 1280 1024 ) #bounds: #(#Rectangle 395 372 791 482 ) ) #component: #(#SpecCollection #collection: #(#(#LabelSpec #layout: #(#AlignmentOrigin 0 0.5 10 0 0.5 0 ) #name: #label #label: 'label' ) #(#InputFieldSpec #layout: #(#LayoutFrame 16 0 -77 1 -13 1 -53 1 ) #name: #input #model: #input #type: #string ) #(#ActionButtonSpec #layout: #(#LayoutFrame -77 0.5 -40 1 -10 0.5 -10 1 ) #model: #accept #label: 'OK' #isDefault: true ) #(#ActionButtonSpec #layout: #(#LayoutFrame 10 0.5 -40 1 77 0.5 -10 1 ) #model: #cancel #label: 'Cancel' #defaultable: true ) ) ) )
```