VRGOODIES

This document describes a set of user interface tools that have been developed in order to enhance the productivity of the VisualWorks/VisualWave developer. These tools fall into the following categories:

- UI Building enhancements
    - Form Generator
    - UIDefiner upgrade
- UI Framework enhancements
    - Subclasses of ApplicationModel that provide additional functionality

The Form Generator

This tool automates the production of stock user interface forms.

Its usage pattern is simple; a menu item in the class pane of the browser summons up the tool on the selected class. The tool comes up with the name of the selected class shown, a proposed Form class name, and a proposed Form class category name. All of these fields are editable. For the chosen class, all instance variables will appear in the leftmost list box. This box is a multi-selection list box, while the rightmost one is single selection. Selected instance variables will be moved to the rightmost list box by using the >> button. Variables may be moved back to the leftmost list box by using the << button. Variables in the rightmost list box will be used to create the form.

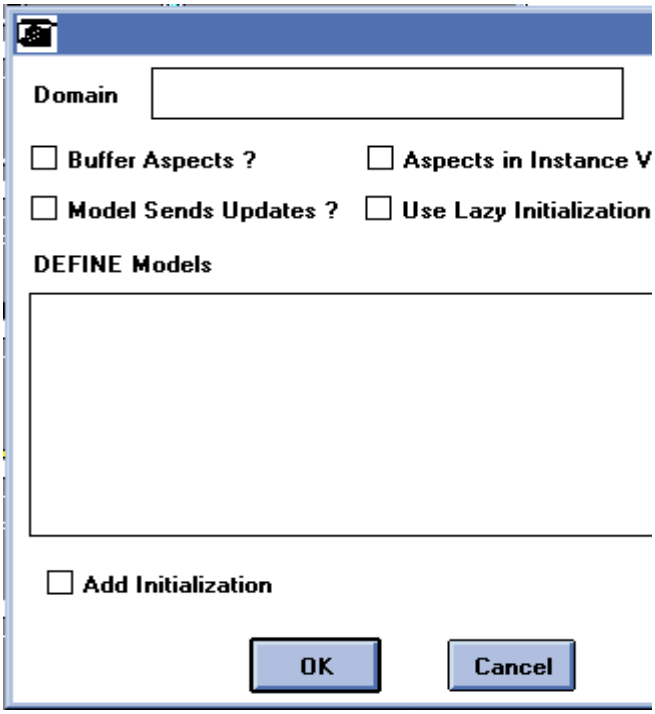
Below the rightmost list box are the widget types supported by the tool. These are as follows:

String	An input box widget
Text	A Text Editor widget
Password	An input box widget, formatted for passwords
Time	An input box widget, formatted for Time objects
FixedPoint	An input box widget, formatted for decimal numbers
Collection	A List box widget
Radio Buttons	One or more radio buttons
Symbol	An input box widget, formatted for symbols
Number	An input box widget, formatted for numbers
Date	An input box widget, formatted for dates
Timestamp	An input box widget, formatted for time stamps
Boolean	A checkbox widget
Subcanvas	An embedded subcanvas
Menu Button	A menu button

Note that the formatting options vary by the selection; for all input field widgets, the menu button below the leftmost list box will display a menu of appropriate formats (as in the standard Property Tool). For other widgets, this part of the UI will change as noted below. Note that selecting a type or format will change the widget information for the selected variable immediately.

Collection	Menu is disabled
Text	Menu is disabled
Radio Buttons	Radio button editor
Menu Buttons	Menu Button Editor
Subcanvas	Subcanvas Editor

Radio Button/Menu Button Editor



Add
Add #select value to the list.

Remove
Remove #select value from the list.

Save
Save information on radio button or menu button. This editor is used for both radio button and menu button editing. Edit values are not saved until the #save button has been used.

Aspect
Name of the value model that will hold the selected value.

Select
The name of one of the options.

ListBox
List of current options that will be used to build the radio buttons or menu.

Subcanvas Editor

Canvas Class
Name of the ApplicationModel subclass to embed.

Canvas Spec
Name of the window specification to embed.

Save Info
Subcanvas information is not saved until this button is pressed.

Generating the interface

In order to generate an interface, the user must set at least the following information:

IFC Class
The class that is to be generated. If <use extended framework> is selected, then this class will be a subclass of VRDetailForm. Otherwise, it will be a subclass of ApplicationModel. If <tabular ?> is selected, then a table view will be generated. In this case, the class will be a subclass of VRTableForm and will be ready for usage as a tabular interface. For usage details on the extended framework, please see the associated documentation.

Category
The category that the class is to be stored in.

Form Generator: Tool Settings

The form generator has a few user settable options, seen in the screen above. These are detailed below:

Input Field Width
The width (in pixels) of an input field. This affects fields that are of one of the following types: String, Password, FixedPoint, Time, Symbol, Number, Date, Timestamp, MenuButton.

InputField Height
The height (in pixels) of an input field. This affects fields that are of one of the following types: String, Password, FixedPoint, Time, Symbol, Number, Date, Timestamp, Menu Button

Label/Input Field Separation
The number of pixels between a label (auto generated) and the input field.

Widget Vertical Separation
The number of pixels (vertically) between widgets.

Large Widget Height
The height (in pixels) of an input field. This affects fields that are of one of the following types: ListBox, Text, Subcanvas.

Large Widget Width
The height (in pixels) of an input field. This affects fields that are of one of the following types: ListBox, Text, Subcanvas.

Widget Indentation
Indentation (in pixels) from the left.

Enhanced UIDefiner

The Enhanced UIDefiner more properly supports the existing GUI painting tools. The definer auto-generates all notification and validation methods that have been defined in the Property Tool; notification methods answer <self>, and validation methods answer <true>. In addition, many new features have been added.

The UIDefiner has the following fields:

Domain
The name of the domain model class that the user interface should be hooked up to. If this is filled in, then adaptor code will be generated to hook the user interface to the domain. If it is left blank, the toll will generate code in the same fashion that it has in previous releases of VisualWorks.

Buffer Aspects
If this is checked, then BufferedValueHolders (or AspectAdaptors) will be generated.

Model Sends Updates
If this is checked, the generated code will assume that the domain model sends change notification messages for it's variables.

Aspects in Instance Variables
If this is checked, then all adaptors will be stored in instance variables rather than being returned from aspect methods.

Use Lazy Initialization
If this is checked, then adaptors will be returned from aspect methods. Otherwise, they will be set during initialization.

Add Initialization
If checked, variables will be preset with appropriate values based on type information. This setting is ignored if a domain model is being hooked up.

Note that asking for no instance variables **and** non-lazy initialization is incompatible; the tool will not allow this option. If a domain model is being hooked up, there is an additional check done. If that domain model exists, then no code is generated for the model. If that model does not exist, then it is generated as follows:

1. The class will be a subclass of Model
2. A #new method will be generated that sends #initialize
3. An #initialize method will be generated that sets the variables in the domain based on type information acquired from the GUI tools.
4. Accessing methods (which respond with change information) will be generated.

If non-lazy initialization is chosen, then the following methods will be defined in the user interface class (where MyModel is the name of the domain model):

```
initialize
   "UIDefiner defined this method.
   Do NOT modify this method; place custom
   code in <initializeApplication>"

   super initialize.
   self model: MyModel new asValue.
   self initializeAspects.
   self initializeApplication.

InitializeAspects
   "Preset all aspects.  UIDefiner will
   overwrite this method"
```



```
      fieldSeparator:= ((AspectAdaptor subjectChannel: self model
         sendsUpdates: true)
         forAspect: #myModelAspect).

   initializeApplication
      "UIDefiner defined this method. UIDefiner
      will not recreate this method, so all
      custom initialization code should be placed here."
```

The #initialize and #initializeAspects methods will always be redefined by the UIDefiner, so customization should be placed in the #initializeApplication method. Note that when the user interface is being hooked up to a domain model, all instance variables defined will be hooked to the domain model.

The domain model (when automatically defined) is defined by a new class, ModelDefiner.

User Interface Frameworks

In addition to the tool level enhancements, a number of ApplicationModel level framework enhancements have been added as well. These classes are briefly described below.

CommonApplicationModel

Subclass of ApplicationModel

This class contains all the 'convenience' protocol that is normally created in an abstract subclass of ApplicationModel. For instance, instead of the following message send to enable a widget:

```
(self builder componentAt: #widgetID) enable
```

this class adds protocol that allows:

```
self enable: #widgetID
```

This makes the developer's task easier, and adds to their general level of productivity. All such common protocol is included in this class. In the documentation below, protocol names are in italics, method names are bolded.

single-widget management

autoAccept: aSymbol
Turn on auto accept.

autoAcceptOff: aSymbol
Turn on auto accept.

backgroundColor: aSymbol to: aColorValue
Set the color.

beInvisible: aSymbol
Invisible the widget.

beVisible: aSymbol
Invisible the widget.

changeListFont: aSymbol
Change the font of a listView object.

changeListFont: aSymbol to: aFontName
Change the font of a listView object.

component: aSymbol
Return the wrapper.

controller: aSymbol
Return the controller of the widget.

disable: aSymbol
Disable the widget.

enable: aSymbol
Enable the widget.

foregroundColor: aSymbol to: aColorValue
Set the color.

hide: aSymbol
Disable, make invisible the widget.

inputLimit: aSymbol
Return the input limit.

label: aSymbol
Answer the current label as a string.

label: aSymbol with: aComposedText
Label a widget - assume input is ready.

label: aSymbol withImage: anImage
Label a widget - assume an image.

label: aSymbol withString: aString
Label a widget - assume all bold - if multiline, auto expands.

labelWidget: aSymbol withImage: anImage
Label a widget - assume an image.

labelWidget: aSymbol withString: aString
Label a widget - assume all bold - if multiline, auto expands.

modelFor: widgetId
Answer the valueModel for the widget.

move: aSymbol by: anOffset
Move widget by an offset.

move: aSymbol to: aPoint
Move widget to a point.

replaceControllerOf: aSymbol with: aController
Replace the controller for widget belonging to the componnet whose id is aSymbol with aController. Make sure the new controller has the old controller's menu and performer. Also make sure the new controller references the keyboard processor.

replaceWidgetOf: aSymbol with: aWidget
Replace the widget.

selectionBackgroundColor: aSymbol to: aColorValue
Set the color.

selectionForegroundColor: aSymbol to: aColorValue
Set the color.

show: aSymbol
Enable, make visible the widget.

takeFocus: aSymbol
Have component grab focus.

turnOff: aSymbol
Turn off the widget.

turnOn: aSymbol
Turn on the widget.

widget: aSymbol
Return the widget.

multi-widget-management

autoAcceptAll: anArray
Turn on auto accept.

backgroundAll: anArray to: aColorValue
Change the background colors.

beInvisibleAll: anArray
Make all invisible.

beVisibleAll: anArray
Make all visible.

disableAll: anArray
Disable the object(s).

enableAll: anArray
Enable the object(s).

foregroundAll: anArray to: aColorValue
Change the foreground colors.

hideAll: anArray
Hide the object(s).

labelAll: anArray withImage: anImage
Label all with same image.

labelAll: anArray withString: anImage
Label all with same string.

moveAll: anArray by: aPoint
Offset all the object(s).

selectionBackgroundColorAll: anArray to: aColorValue
Change the foreground colors.

selectionForegroundColorAll: anArray to: aColorValue
Change the foreground color.

showAll: anArray
Show the object(s).

turnOffAll: anArray
TurnOff the object(s).

turnOnAll: anArray
TurnOn the object(s).

interface opening

openAt: aPoint with: aSymbol in: aRectangle
Open the interface at a particular location - assume a main window.

openAt: aPoint with: aSymbol in: aRectangle ofType: aType

Open the interface at a particular location.

scaleRect: aRect
Stub - user may intervene to scale rectangle, perhaps based on screen size.

aspects

adapt: anObject using: aSymbol
Assumes you want updates.

bufferForAspect: aSymbol trigger: aValueModel
Assumes you get updates.

bufferNoUpdateForAspect: aSymbol trigger: aValueModel
Assumes you don't get updates.

modelForAspect: aSymbol
Assumes you want updates.

modelNoUpdateForAspect: aSymbol
Assumes you don't get updates.

modelValue
Answers the model for this interface.

dependents access

registerModel: aClass
Register self with domain.

registerModel: aClass asDependent: aBoolean
Register self with domain.

setModel: aModel
Assume the use of 2.x aspect paths: so, model is a ValueModel.

keyboard access

keyboardHook
Return the main keyboard hook.

keyboardHook: aBlock
Install a new keyboard hook, return the old one.

keyboardProcessor
Return the keyboard processor of the window.

api

doFileOpen: aFileString
Subclass responsibility.

doFileSave: aFileString
Subclass responsibility.

common dialogs

openFile
Answer a filename or nil using class CommonFileSelectionDialog.

saveFile
Answer a filename or nil using class CommonFileSelectionDialog.

selectDirectory
Answer a directory name or nil using class CommonDirectorySelectionDialog.

selectFromList: aList
Answer a selection from the list or nil.

selectionsFromList: aList
Answer the selections from the list or #().

warnCritical: aMessage
Dialog warning box.

warnInformative: aMessage
Dialog warning box.

warnNormal: aMessage
Dialog warning box.

warnQuery: aMessage
Dialog warning box, with confirmation requested.

model utilities

aspectAdaptorFor: anAspect
Answer a new adaptor for this aspect.

bufferedValueHolderOn: aModel
Answer a new bvh for this aspect.

bufferedValueModelIds
By default, the application does no buffering of data. Subclasses may wish to override this by returning an array of widget ids that are buffering user input.

bvhWithAspectAdaptorForAspect: anAspect
Answer a new bvh for this aspect.

ExtendedApplicationModel

Subclass of CommonApplicationModel

This class contains convenience code targeted at standard application behaviors.
It adds three instance variables:

model
The domain model for this application.

trigger
A trigger that may be used by BufferedValueHolders.

dialogBuilder
Holder for dialog builders.

These variables are commonly defined, so this class stands as a placeholder for that. In addition, it contains protocol for wait/cancel dialogs, screen positioning of an interface, and standard release behavior.

accessing

dialogBuilder
Cached builder for dialog box that is raised.

dialogBuilder: aValue

model
My model (normally, a ValueHolder on a model).

model: aValue

trigger
A ValueHolder on a boolean; used by BufferedValueHolders.

trigger: aValue

interface customization

customizeBuiltSpec: aSpec
Make any mods to the spec object here.

customizeSpec: aSpec
Make any mods to the spec (the pre-built description) here.

openInterface: aSymbol withPolicy: aPolicy inSession: anApplicationContext
Open the ApplicationModel's user interface, using the specification named and the given look policy and application context.

postBuildWith: bldr
Register to grab window events.

window-management

expandWindow
Deiconify window.

hideWindow
Unmap the window.

iconifyWindow
Iconify window.

showWindow
Map the window.

window
Return the window.

private

copyVarsFrom: oldObject to: newObject
Copy contents of inst vars where they match.

initialize-release

release
Include a default release behavior that will forward release to model.

release: aCollection
Assume that we want to release a lot of objects.

aspect-management

optimizeInterestIn: aValueModel using: aSelector for: anObject
Register interest in a value model, with the assumption that the selector is unary This avoids overhead of symbol parsing normally done in #onChangeSend:to:.

screen printing

captureScreen
Capture and return the current window as an image.

printScreen
Print the active window - rely on Host OS for error message.

screenToClipboard
Copy the current window to the system clipboard *other services.*

executeWithCancel: aBlock
Execute a block with cancel protection.

executeWithCancel: aBlock with: aMessage
Execute a block with cancel protection.

executeWithCancel: aBlock with: aMessage at: aPriority
Execute a block with cancel protection.

executeWithWait: aBlock
Execute a block with wait dialog.

executeWithWait: aBlock with: aMessage
Execute a block with cancel protection.

executeWithWait: aBlock with: aMessage at: aPriority
Execute a block with cancel protection *platform* return the platform name.

screenSize
Return a point, where x is the width, y is the height.

events

noticeOfWindowClose: aWindow
If my window is closing, invoke my release behavior, which will also release model.

drag-drop support

getElementIndexOfTarget: mousePoint for: targetController
Given context, controller, grab the element that was dropped on.

startDrag: dragEvent with: dragDictionary for: aController
Set up the drag drop event.

error reporting

messageFor: exception
Answer the normal error string; override for more complex handling.

reportError: exception
Subclasses might wish to override; this method throws a dialog.

actions

accept
Trigger an accept to any bufferedValueHolders.

cancel
Cancel bufferedValueHolder accept.

shutdown
Discard any buffered values and close.

ServiceApplicationModel

Subclass of ExtendedApplicationModel

This subclass of ExtendedApplicationModel factors out the 'resource' pieces of the framework. Added at this level are the 'child window' and simple printing support.

Window Services

We can get 'MDI like' services by registering windows as child. Child windows get the collapse, close, and expand events forwarded to them. Other window messages have been shortened. Check in the protocols window-management and child-management.

Printing support

This support utilizes the document class. Five methods are implemented, as follows:

print
The entry point. Creates an instance of FormattingStream (a subclass of textStream), then sends the message #print: to itself.

print: aStream
This is a subclass responsibility. This method should fill the stream with text.

actuallyPrint
If the variable shouldPrint is true (by default, it is - subclasses may wish dialog control to change it), this method will create an print the document. It may be modified as follows:

font: whatever is returned by the message #styleType
page: whatever is returned by documentType (#portrait by default)
footer: whatever is returned by #getFooter

child management

addChildModel: anAppModel
Add an app model to our list.

hasChildren
Answer a boolean.

hasParent
Answer a boolean.

removeChildModel: anAppModel
Remove an app model to our list.

initialize-release

initialize
Set up child class support.

release
Release child class support.

printing-support

canPrint
Application specific test.

characterWidth: aStyleName
Answer the width of a character, in case user wishes to format.

documentType
Answer #landscape or #portrait.

getFooter
Answer an empty footer - subclasses may override to do something special.

getHeader
Answer an empty footer - subclasses may override to do something special.

headerSize
Answer the pixel size for the header.

styleType
Answer a known font: #fixed is our default.

printing

actuallyPrint: aStream
Print the contents of the stream- create a document, send to printer. Due to win32s bug, default choice (landscape) should match choice in Windows print setup. subclasses should override for custom behavior.

actuallySave: aStream on: aString
Subclasses may override for custom behavior.

print: aStream to: aStringOrNil
Default printing. subclasses should override #producePrintStream:.

printToFile
Print to a file determined by the dialog.

printToFile: aFileString
Print to a stream- answer the stream.

printToPrinter
Print the stream.

producePrintStream: aStream
Subclasses should override to fill the stream.

dependency

dependOn: aModel
Add self as a dependent of a model.

eventFrom: aValueModel sends: aSymbol
Add a change event.

accessing

bufferedValueModelIds
By default, the application does no buffering of data. Subclasses may wish to override this by returning an array of widget ids that are buffering user input.

depRegistry
Registry of dependents; developer managed.

depRegistry: aValue

extChildren
Child windows (list).

extChildren: aValue

extOwner
Parent window.

extOwner: aValue

printStream
Stream to print on.

printStream: aValue

shouldPrint
Boolean; can we print from this form?

shouldPrint: aValue

testing

isEditing
Is this an editing form (a boolean).

okToChangeModel
True if model can be changed, false otherwise.

events

noticeOfChildWindowClose: anAppModel
A child window has closed; execute any appropriate behavior.

noticeOfWindowClose: aWindow
Fire #release, where all cleanup should be (VisualWave compatibility).

requestForWindowClose

VRApplicationModel

Subclass of ServiceApplicationModel

This class adds a 'UI Registry' concept to VisualWorks.

To use this, there are two APIs:

answerFromRegistry: aClassName

openFromRegistry: aClassName

In both cases, the application returned is cached in a dictionary for later use. This is useful for presetting a UI, or for ading behavior that merely hides closed windows instead of destroying them. There is a fair amount of support code included in this class to support this, including the proper setting of instance variables for follow on user interfaces.
In a VisualWave environment, a hyperlink framework has been added as well. This code allows VisualWave interfaces to treat standard URLs as user interface button presses, thus allowing for the display of a 'standard' web interface.

private

insertToRegistry: aSymbol
Create a new instance, with the same registry.

openNew: appModel spec: spec
Open a new instance that is not in the registry.

openOld: appModel spec: spec
Open an existing instance which is in the registry.

actions

submit
Convenience method.

api-error reporting

reportDBError: aString
Report the db error.

reportError: aString
Report the error.

reportListError: aString onList: aList
Report the list of errors.

reportNonModalDBError: aString
Report the db error, modally.

reportNonModalError: aString
Report the error, modally.

reportNonModalListError: aString onList: aList
Report the list of errors, modally.

api-ifc control

closeAllPagesInMySession
Loop through and close all open pages in my session. If I'm in the web world just ask my session for its controllers. If I'm in the screen world, don't close all windows in the session (which is the global control manager), just close the windows for the applications in my registry.

interface opening

postOpenWith: aBuilder
After the interface has been opened (but not yet sent to the browser if in VisualWave) I now can create create any hyperlinks on the page that map back to me. Modify the html text widgets to hold HREF with the appropriate URLs.

api-registry

answerFromRegistry: aSymbol
Assume the symbol to be a class name, make sure to pass on the registry.

clearRegistry
Clear the registry.

findAppInRegistry: appModel
Return the instance of the app model found in the registry - nil if none found.

findAppInRegistryByKey: aSymbol
Return the instance of the app model found in the registry - nil if none found home answer the 'top' page of the registry - subclasses must implement if they expect this to work.

openFromRegistry: aSymbol
Assume the symbol to be a class name.

openFromRegistry: aSymbol using: aSpecName
Assume the symbol to be a class name.

removeFromRegistry: appModel
Remove appModel from registry.

removeFromRegistryByKey: aSymbol
Remove appModel from registry.

events

handleClientPullEvent
By default, do nothing. Subclasses may wish to implement. This is VisualWave specific.

initialize-release

initialize
Set up the registry.

initRegistry
Actually set up registry.

release
Does not call #releaseRegistry, as only top level appModel can safely do so.

releaseRegistry
Release the registry. top level appModel should do this.

accessing

creationFlag
Either #new or #existing. Internally used for caching.

enclosingFrameset
Only used in VisualWave. In that case, holds the frame appModel that is framing me.

enclosingFrameset: aValue

registry
Answer the registry object.

registry: aValue
Set the registry object.

selectorsEligibleFromHyperlinks
The set of selectors that can be used for links.

selectorsEligibleFromHyperlinks: aCollection

shouldCache
If true, window closure will hide and cache the window. Else just close.

shouldCache: aValue

testing

isDataModel
Is this a dataModel class (subclass of ExtendedDataModel).

isDBAppModel
Is this a subclass of VRDBApplicationModel (used to determine which data to pass down).

isFramed
Am I being framed?

isRefreshEvent: submitController
Answer true if client pull event.

web-dialog-opening

dialogWarn: aString
Dialogs for pre 2.0 Wave apps.

aspects

errorText
Answer error text.

submitting

submitFrom: submitController toComponents: componentCollection
First, check if this is a client pull event. Next, handle any submissions caused by clicking on a hyperlink that maps to an action for an application to take. Finally, proceed with the default action.

private-hyperlink response

currentlyDisplayedSubApplications
Return a collection of any currently displayed sub applications. Subclasses may wish to override if they have subapplications that need to respond to hyperlink clicks. Only page-level applications receive notification of page submission, so if the page doesn't respond to the hyperlink's associated action, this list will be used to forward any unconsumed hyperlink clicks on to any subapps for possible handling.

findHyperlinkConsumerForMessage: aMessageSelector
Find the intended consumer for the message send of MessageSelector. The consumer could be me or one of my subapplications. Remember, only page-level applications are informed of page submissions, so if the page-level application (self) is not the consumer (indicated by whether or not I understand the message), then determine if one of my subapps is the intended consumer. Assume that the first subapp that says they have registered a web interest in the message is the intended consumer (for this reason, care must be taken that my subapps aren't registering a web interest in the same message selectors).

handleHyperlinkClicksInWebRequest: aWebRequest
Interrogate the URL used to contact VisualWave. If the query string contains a parameter of 'action' this indicates that this contact is the result of the user clicking on a hyperlink that was dynamically generated by the application to map to an action that the application should perform. Be sure to convert the argument for 'action' to a symbol because we couldn't store a symbol in the URL originally and had to convert the message selector to a string. Next find the intended consumer for the action which could be me or one of my subapplications. If a consumer was found ask it to perform the intended action making sure to pass any parameters that were embedded in the URL.

hasWebInterestInSelector: aMessageSelector
This message should fire a hyperlink if true.

haveConsumer: anApplication perform: aMessageSelector via: aWebRequest
Ask anApplication to perform aMessageSelector passing any arguments as needed. Remember, the only thing that can be stored in URLs are strings, so the eventual consumer will need to perform any neccessary argument conversions (for example, if a '3' is really intended to be a 3, convert it).

private-hyperlink creation

hyperlinkForText: aTextString toPerform: aMessageSelector
Convenience.

hyperlinkForText: aTextString toPerform: aMessageSelector andEmphasis: anArrayOfEmphasesOrNil
Convenience.

hyperlinkForText: aTextString toPerform: aMessageSelector withArguments: argumentsOrNil
Convenience.

hyperlinkForText: aTextString toPerform: aMessageSelector withArguments: argumentsOrNil andEmphasis: anArrayOfEmphasesOrNil
Create a hypertext link for aTextString. The HREF for the link will contain an URL that maps back to me (the running application) and a reference to a message to be sent to myself when this URL is accessed. I cannot embed the messageSelector unchanged because of the $#, so convert the symbol to a string. Because of limitations in what can be stored in an URL, each argument must be reduceable to an opaque string (i.e. a string with no embedded whitespace). It is assumed that the eventual receiver of the message will be able to perform any massaging of the strings into the appropriate objects – for example '3' is really 3. Register a web interest in the message selector (in other words, indicate that I respond to this selector from web requests. Since many applications could respond to the same message, we register those selectors that each application has indicated they respond to from the web so that only the application that has registered a web interest will receive the message.

registerWebInterestInSelector: aMessageSelector

setHyperlinks
Populate the specified html text widgets with anchors to hrefs. Subclasses may wish to override this.

VRDBApplicationModel

Subclass of VRApplicationModel

This class adds an instance variable, dataModel. The intention is to support applications that use the ObjectLens in a programmatic fashion. Along with the instance variable, proper release behavior is added. Note that with the registry added in VRApplicationModel, the dataModel presented here will not be released unless the message #releaseDataModel is sent.

accessing

dataModel
The dataModel being held.

dataModel: aValue

initialize-release

releaseDataModel
This should be called by the top level appModel in the chain.

api

answerFromRegistry: aSymbol
Assume the symbol to be a class name, make sure to pass on the registry.
Overriden to make sure dataModel variable is passed on to new entrants

logout
Logout of the database, releasing sessions as well.

testing

isDBAppModel

ExtendedCompositeApplicationModel

Subclass of CompositeApplicationModel

An analog of ExtendedApplicationModel for VisualWave users of framesets. This class merely duplicates all registry code for the frame case.

VRDetailForm

Subclass of VRDBApplicationModel

This class provides protocol that is compliant with VRPagngForm, and VRTableForm. It also provides 'plug-in' behavior for printing and saving domain models (which must implement behavior in order to handle these requests). This is the final abstract subclass of ApplicationModel for detail forms. As such, the Form Generator will generate subclasses of this by default.

interface opening

customizeClient
This message is sent by VRPagingForm when the wrapped form has been paged to.
This interface is intended to act in the same fashion as postBuildWith.

postBuildWith: bldr
Customize self on opening.

initialize-release

initialize

initWith: aModel
Interface for VRPagingForm.

private

returnToParent
Raise parent window if have it.

actions

accept
Perform acceptance based on mode.

add
Adding a new entity.

cancel
Leave actions here to concrete subclasses.

edit
Accepting an edit.

accessing

editMode
Answer current editing mode (either #edit or #view).

editMode: aValue
Set editing mode.

owner
Parent view (the wrapper if there is one).

owner: aValue

parent
Parent view.

parent: aValue
Parent View.

windowLabel

windowLabel: aValue

testing

showEditModeButtons
Answers false; set to true if edit buttons should display.

api

customizeEmbeddedSubcanvas: spec
We are embedded, and subcanvases don’t <do the right thing>. So fix it.

disableEditButtons
Disable edit buttons.

enableEditButtons
Enable edit buttons.

useConstantData: data
Data cached in paging form that may be needed.

VRPagingForm

Subclass of VRDBApplicationModel

This is a ‘wrapper’ class. It is intended to wrap an existing ‘detail’ form. As such, it assumes the presence of protocol provided in class VRDetailForm. This class provides ‘paging’ behavior to detail forms via a set of VCR type controls.

actions

firstObject
Page to the first object in my internal list.

lastObject
Page to the last object in my internal list.

nextObject
Page to the next object in my internal list.

prevObject
Page to the previous object in my internal list.

events

addNewItem: anItem
Add an item to the list.

changedView
Changed the model underneath.

initialize-release

initialize
Set up internal dependencies.

initWith: aList on: aContainedForm
Set up for the new list; cache the detail form.

release
Release all dependencies that have been set up.

accessing

bindings

bindings: aValue

constantData
This is a cache (developer dependent) that will be passed between detail forms.

constantData: aValue

dataView
This is the holder for the current detail form being displayed.

dataView: aValue

modelSL
The model holder.

modelSL: aValue

newLabel

newLabel: aValue

noHideButton

windowLabel

windowLabel: aValue

updating

update: anAspect with: aValue from: aModel
Handle internally set dependency events.

interface opening

customizeBuiltSpec: aSpec
Change the spec to use the actual embedded canvas.

hideCloseButton
If true, hide the close button on the wrapper form.

postBuildWith: bldr
Customize UI with any cached window labels (inst var windowLabel).

testing

hasDynamicBindings
Set to true; this is for use by the builder in the subcanvas construction process. In ApplicationModel, this answers false.

private

atBinding: aKey put: aValue

bindingsFor: aKey

setCountLabel: aCount
Set the current count on the wrapping form.

VRTableForm

Subclass of VRDBApplicationModel

This class provides protocol needed by tabular representation of data (as defined by the Form Generator). It wraps creation of VisualWorks tables in a convenient 'spec' method, allowing for easy modification of displays. Subclasses of this class are plug compatible with VRListForm, and supports VRPagingForm via protocol

initialize-release

initialize

initializeTable: dataList
Set up table. This is the api usually used to set the form.

initializeTable: dataList from: aSpec
Set up table.

accessing

detailForm: aValue
Set the detail form that may be edited.

domainFormClass

domainFormClass: aValue

printHeader
Print api.

printHeader: aValue
Print api.

WebTable
The embedded table.

webTable: aValue

windowLabel
The label to display on the window.

windowLabel: aValue

interface opening

postBuildWith: bldr

private

producePrintStream: aStream
Iterate over the collection, putting a form feed character into it after each element. Note the domain elements must know how to represent themselves on a stream in order for this to work.

actions

add
Add a new entity by popping a detail form.

edit
Edit selected object with a detail form.

editAll
Bring up a paging form on the list.

print
Print the objects, with a form feed between each.
NOTE: Objects in collection must respond to the #printOntoStream: message.

remove
Remove selected element.

save
Save the collection of objects; this follows the same protocol as print, But first pop a dialog (CommonFilesSelectionDialog) to select the save file.

VRListForm

Subclass of VRDBApplicationModel

This class provides protocol for displaying lists of information. It must be given a detail form for such objects in order to present editors. As such, it expects the protocol presented by VRDetailForm and VRPagingForm. In order support printing and saving of objects, the domain objects must implement appropriate protocol. This class is plug compatible with VRTableForm.

actions

add
Add a new entity by popping a detail form.

edit
Edit selected object.

editAll
Bring up a paging form on the list.

print
Print the objects, with a form feed between each. NOTE: Objects in collection must respond to the #printOntoStream: message.

remove
Remove selected entity from the list.

save
Save to a formatted file. NOTE: Objects in collection must respond to the #printOntoStream: message.

aspects

dataList
The internal list of objects.

accessing

dataList: aValue

detailForm: aFormClassSymbol

domainFormClass

domainFormClass: aValue

offList

printHeader
Label for top of printed pages.

printHeader: aValue

windowLabel
Label for top of windows.

windowLabel: aValue

initialize-release

initialize

initializeTable: data
Makes it easy to sub a list form for a table form.

on: aList detailForm: detailClass
Initialize the form. This is the api usually used.

interface opening

postBuildWith: bldr

api

addDomainObject: anObject
Add an object to the list.

turnOff: aList
Make the list un-editable.

turnOn: aList
Make the list editable

printing

producePrintStream: aStream
Iterate over the collection, putting a form feed character into it after each element.

Extended ObjectLens Framework

Included in this framework are a set of classes that extend LensAppicationModel. In general, these classes make it easier to use the ObjectLens programmatically. As provided, most of the protocol useful for programmatic usage is held in LensSession. This framework extracts a subset of the most useful protocol (while adding useful query protocol as well).

ExtendedDataModel

Subclass of LensApplicationModel

This class provides convenience protocol for programmatic usage of the ObjectLens. It provides convenient protocol for interaction with the Lens by front-ending most of the protocol that exists in class LensSession.

transactions

abortTransaction
End a transaction.

beginTransaction
Start a transaction.

endTransaction
Wrap up.

okTransaction
End a transaction.

private

authenticate
Override the default to avoid the dialog. Developer must have set username, password into our instance variables.

executeWithProtection: aBlock
Execute the block, looking for SQL errors.

messageFor: exception
Answer the string error message in the exception.

notifyUserOfLensException: anException
Catch lens exceptions and report them.

postPrepareSample: aSampleObject
Protocol for QBE queries.

prepareSample: aSampleObject
Prepare a sample object for QBE queries.

reportError: exception
Subclasses might wish to override; here, pop a dialog.

initialize-release

initialize

release
Release all dbms related resources; rollback any uncommitted changes.

accessing

fail
Answer the fail message that has been cached.

fail: aValue
Cache a failure message.

pass
The valueHolder on the password to use for login

pass: aValue

reason
Cache for failure to login reason.

saveSample
Cache a sample object for use in QBE queries.

transactionState
Answer the transaction state.

user
The valueHolder on the username to use for login.

user: aValue

api

addAll: aCollection
Add items to the database.

addObject: anObject
Add object to the database.

login: username password: password
Login to the database.

removeAll: aCollection
Remove items from the database.

removeObject: anObject
Remove object from the database.

updateAll: aCollection
Update objects into database.

updateObject: anObject
Update object to the database.

api-query

doAllQuery: classNameSymbol forTable: tableNameString
Execute a 'select *' for the named table. Note that a bind class must exist in the dataModel.

doEXDICommand: sqlString
Direct EXDI access for SQL; subclasses must customize to handle query results.

doSQLQuery: aString forClass: classNameSymbol
Create a lensQuery for an arbitrary SQL string.

performQBE: aSymbol using: aSampleObject
Perform a qbe using sample object.

performQuery: aSymbol
Execute the named query and answer a collection of objects.

testing

isDataModel
Answer true.

isDBAppModel
Answer false.

AbstractVRDataModel

Subclass of ExtendedDataModel

This class extends its parent in one way – instead of reporting errors via dialog box, it caches them.

reportError: exception
Cache the error string in the <reason> instance variable.

