Developing Windows Applications with Visual Framework Tutorials

Visual LANSA Frameworks (VL Framework) provide a powerful application framework for rapidly prototyping and developing both Windows and Web applications.

The exercises in this tutorial introduce the fundamental skills needed to develop Windows applications using the Visual LANSA Framework.

The exercises in this tutorial are:

- VFW005 Basic Windows Controls
- VFW010 A Tab Folder Framework
- VFW020 Execute a Visual LANSA Framework Application
- VFW030 Create a Prototype
- VFW040 Snap in Real Filters
- VFW042 Snap in a Real Command Handler
- VFW044 Add Instance List Columns
- VFW050 Basic Combo Box Processing
- VFW052 Build a Working List of Selected Items
- VFW054 Edit Text in a Memo / Edit Box
- VFW056 Process a List in Sorted Order
- VFW060 Using a Tree View
- VFW062 A Tree View with Columns
- VFW070 Create a Reusable Part Object
- VFW072 Create a Department Dropdown Reusable
- VFW074 Create a Compound Reusable Part
- VFW080 Using an Explorer Component
- VFW082 Toolbars, Menus and Pop–up Menus
- VFW084 A Business Object Browser and Detail
- VFW090 Field Visualizations

- VFW100 Define a Parent/Child Instance List
- VFW102 Field Visualizations in a Grid
- VFW104 Simple Keyed Collections
- VFW106 Using a List Collection
- VFW110 Simple Drag and Drop
- VFW112 Drag and Drop between Components
- VFW120 Using Hidden Commands
- VFW122 Launching a VLF Window
- VFW124 Using Business Object SubTypes
- VFW126 Using Space Objects (Optional)
- VFW130 Set up the VL Framework for Client/Server

Edition Date August 11 2014 © LANSA

About the Exercises

Who Should Use the Exercises?

These exercises have been written for LANSA developers who have completed the Visual LANSA Fundamentals training. Basic Repository and RDML programming skills are required. You should have completed the Visual LANSA User Interface Tutorial, Editor Tutorial, Repository Tutorial and the RDML Programming using Forms Tutorial before starting these exercises.

These exercises will expand upon the basic form programming skills learned in the Fundamentals tutorials.

How Many Developers Can Use the Exercises?

Except if you are using a trial version of Visual LANSA without a development license, there is no limit to the number of developers who may use the exercises at the same time in the same partition. However, it is important that each developer has a unique identifier for their own work as noted below.

If you have a **development license**:

• To allow for more than one developer to do the exercises in the same partition, all LANSA object names are prefixed with iii which represents a unique three character code assigned to each exercise user. For example, if your name is John David Smith you can use the characters JDS. When asked to create a form named iiiVFW01, you will create a form named JDSVFW01.

If you are completing more a LANSA exercise more than once, and you are asked to create another iiiVFW01 form, you can simply use a different set of characters for iii.

• Always remember to replace iii with your unique 3 characters when creating objects in LANSA. You will not always be reminded to make this substitution.

If you are using LANSA in **trial mode**:

• Only one developer can do the exercises in the same partition because object names must be unique. In trial mode, only a prefix of iii=DEM is allowed and components must be named DEMCOM01 to DEMCOM10. If objects DEMCOM01 to DEMCOM10 already exist, they must be deleted from the repository before you start these exercises.

Install the Tutorial Files

You may wish to install a separate Visual LANSA System for training. You can install an Independent Visual LANSA Workstation on a PC for training and then uninstall it once training is complete. If you are using Visual LANSA in trial mode, this is the recommended approach.

These exercises assume that you have not customized the editor interface unless instructed in the exercise. If you have made customizations, some instructions and sample screen images may not exactly match your development environment.

What Partition Should I Use?

Any partition may be used for training purposes. In these exercises it is assumed that you are using a DEM partition. The partition must be defined as:

- a multilingual partition with
- Long Names enabled and
- be RDMLX-enabled.

The partition must be initialized to load the Visual LANSA Frameworks and Demonstration Material into the partition you will be using as shown in the following screen capture:

🔄 Vist	ual LANSA Logon	23
User I	Partition Initialization	
Passw Us	Mandatory Partition Initialization	
Partiti	Visual LANSA Framework	
SYS	Enable for the Web	
TRN	LANSA Client field and file definitions	
	Demonstration material	
	Run Demonstration	
	OK Show Last Log Messages Cancel Help	
Ok	System Init Partition Init Messages Cancel	lelp

Refer to the Partition Initialization options in the *Visual LANSA Administration Guide* for more details.

Following is an example of a partition's settings (partition DEM) when opened in the Visual LANSA Editor for a stand-alone Visual LANSA install.



These exercises operate the best in a brand new installation using the DEM partition.

Before You Begin These Exercises

LANSA V13 introduces Long Names support in the Repository and it is assumed that Long Names are enabled in the partition you will be using.

- With Long Names enabled, objects have two names, a Long Name and an Identifier (also referred to as the Short Name or Object Identifier).
- With Long Names enabled, when objects such as fields, files, forms and reusable parts are created, the Long Name must be unique within the partition and may not be the same as an existing Identifier.
- A Long Name may be up to 256 characters long and may be letters and numbers with no embedded blanks. Long Names are not case sensitive, so EMPNO, EmpNo and Empno are all the same.
- An Identifier may be up to 10 characters long and may contain letters and numbers and some special characters for some objects, but these are not recommended. Field Identifiers are limited to 9 characters.
- When an object is created using a Long Name, LANSA will assign an

Identifier. As you create objects, you may assign an Identifier (as long as it is unique within the partition). An Identifier cannot be changed once a new object has been saved.

• For more detailed information see *LANSA Object Names* in the *Technical Reference Guide*.

Using Reusable Parts with Long Names in Visual LANSA Frameworks

When you plug-in a reusable part to the Framework:

- An Identifier must be used, since the framework will be loading and unloading this component.
- The recommended procedure is to use the Find dialog, search and find the component required. This will ensure the Identifier is always used.

Filters	Filter S	Settings C	Commands Ena	bled Com	mand Display	Custom Properties	SubTypes	Instance List / Relatio
Identit	fication	Icons F	ilter Snap-in S	ettings				
Stay A	ctive	Defa	ult 💌					
Filter	- Handler							3
Win	dows							
۲	Compo	nent				A		
0	Mock U	b - RAD-PA	D RAD	PAD 9063	501C856D646	2186FF4D41A56059	1C.HTM	1
		Car			10 200 000 0 10			
		3 Find						
		Like Na	ame	IIIVFW)
		Like De	escription					1
								Find
		Name		Descripti	on			1
		IIIVFW0)4	Employe	e Filter by Na	me		
		THVFWU	15	Employe	e Filler By Loo Details CH	cation		
		IIIVEWO)7	Weekly	Report CH			
		IIIVFW1	5	Find Ima	ge			1
		IIIVFW1	6	Employe Salw Cor	e Image Com	mand Handler	_	1000 000 00
mar	mo	at nove we		SalyLa	indenti ri aliti	9	- and a second	

Tips for doing the Exercises

It is recommended that you complete the exercises in sequence.

The first steps in a exercise will provide very precise descriptions of the tasks to be performed. As the steps and course progress, the instructions will become much more general.

Later exercises are designed to use skills from the earlier exercises. Where specific exercises must be done before starting an exercise, the exercise prerequisites are listed in *Before you Begin*.

The exercises are sequenced so that you develop more advanced and complex example applications as the course progresses. For example *VFW001 – Basic Windows Controls* is an optional first exercise designed to establish a basic knowledge of Windows controls. You may not find it necessary to do this exercise.

Note:

- Check off each step in the exercise as you complete it.
- Follow the instructions very carefully.
- Your code may not be an exact match to the examples in these exercises. Do not be concerned if some values do not match exactly. For example, the width of a button is based on the size you create it. Sample code might show a button that has *Width(50)* but your code might have *Width(60)*.

If you have customized the editor, some of the sample screen images may not exactly match your development environment. Do not be concerned if the screens are not an exact match. For example, some tabs may be shown undocked from the editor. Undocked tabs appear as a separate dialog instead of being part of an editor pane. Refer to Dock and Undock a Tab Sheet in the *Visual LANSA User Interface Tutorials* if you need a refresher on these subjects.

Your Feedback

Your feedback regarding these exercises will help us improve the overall quality of the LANSA documentation and training. Please e-mail your comments to lansatraining@lansa.com.au.

VFW005 – Basic Windows Controls

Objectives

• To learn about basic Windows visual control components available with Visual LANSA. These controls enable the user to control and interact with the application.

Basic Windows Control:	5	
Options	Member	Basic Windows Contr 23
HomeOffice		My first form
Enter the Options		ОК

To achieve these objectives, you will complete the following:

Step 1. Create a Form

Step 2. Add Controls to a Form shows you how to add simple examples of the following controls to a form:

Check Box Push Button

Edit Box Radio Buttons

Group Box Static

Label Status Bar

Panel Track Bar

Progress Bar

Step 3. Create Tab Folder Form to learn how the tab folders are used.

Before You Begin

To complete these exercises, you should have completed the *LANSA Fundamentals* workshop.

Step 1. Create a Form

1. To create a new form, from the *File* menu, select *New / Form* and then select *Basic Form*.

ile				
Save	Common			
* Close				\$
Open	Field	File	Form	Reusable Part
Last Opened	_		2	
New	Function	Web	Weblet	Multilingual
Save All		Application		Variable

The New Form dialog opens:

	New Form		
Name	iiiBasicWinCtrls		Create
Description	Basic Windows Controls		Cancel
Framework	Personnel & Payroll (HUMAN RESOURCES)	*	Cancer
Group		*	
Identifier	IIIBAS_1		
Enabled For RDMLX	✓		

- 2. Add the form's details:
- Name **iiiBasicWinCtrls** where iii are your course assigned initials. If you are using iii=DEM, your component must be named DEMCOM01 as described in *How Many Developers Can Use the Exercises*? in About the Exercises.

Description Basic Windows Controls

Framework Select from the dropdown list, in this example it's *Personal & Payroll*. When creating new form or reusable part, you should select a suitable *Framework* for it to belong to. Components are grouped on the *Repository* tab under *Organizers / Frameworks*:

Organizers	
🖻 🛶 Database Diagrams	
4 👿 Frameworks	
ADMINISTRATION - Ad	Administration
ADMINISTRATION1 - A	Administration Comp
COMMON - Common	Common
CONTROLS - Control Co	Control Components
CUSTOMER SERVICES - C	Customer Service Co
EXECUTIVE INFORMATN	Executive Information
FINANCIAL CONTROL	Budgeting & Accounti
4 📷 HUMAN RESOURCES - P	Personnel & Payroll
iiiDspEmpDocs	Display Employee Doc
VB_ARROWB	Bitmap - Arrow backw
VB_ARROWC	Bitmap - Arrow curved
VB ARROWF	Bitmap - Arrow forwar

Step 2. Add Controls to a Form

Push Button

A push button allows a user to perform an action.

🛕 Test

A push button can have a bitmap or a text Caption or both.

Push buttons have events such as *Click*, *Got Focus*, *Lost Focus*, *Start Drag*, *Drag Over* etc.

1. The *Design* tab should currently be visible. This is the default when a new form is created. On the *Home* ribbon, select *Controls* from the *Views* menu.



- 2. Drag and drop a *Push Button* onto the form. Note that the new component is automatically named PHBN_1.
- 3. With the push button selected, select the *Details* tab and change its *Caption* property to **Test**.
- 4. To add a *Click* event, ensure that the push button is selected and select *Events* on the *Details* tab. Double click on the *Click* event to create a click event.

Alternatively, right click on the push button and select *Events / Click Event*. In this case the *Source* tab will be displayed, and the editor is positioned at the new event routine.

5. In the push button click event, add code to display a message in a *Message Box*.

Your code should look like the following:

Evtroutine Handling(#PHBN_1.Click) Use Builtin(message_box_show) With_Args(*Default *Default *Default

6. Compile the form using the *Compile* button on the *Home* ribbon:



7. Run the form, by selecting the *Runtime* button on the *Home* ribbon and select the *Execute* button:



8. A click on the push button should display a message box:

Basic windows Con	trols
Test	Basic windows Contr 83
	My first form
	ОК

- 9. Close your form.
- 10. In the *Design* view, select the push button and use the *Details* tab / *Properties* sheet to add an image to the push button.
- 11. Select the VB_MAGLAS image using the ellipsis button for the Image property:

DragStyle	None							
Elipses	None							
Enabled	True	Rep	Repository Find					
Height	25	E						
Hint		E Bi	tmap					
HintShow	True	Name		Description	[
HintShowOfParent	True		LVLIST	Bitmap - List view list				
Image	*NULL	() E V	LVSICO	Bitmap - List view small icon				
ImagePosition	BeforeCaption		3_MAGLAS	Bitmap - Magnifying glass				
Left	21	E VI	MODEM	Bitmap - Modern				
Menu	"NULL	= VI	_NETWDR.	Bitmap - Network drive				
MenuPosition	Bottom	E V	3_NEW	Bitmap - New				
MenuSeparator	False		3_NEWDOC	Bitmap - New document				
ModalResult	None		3_OPEN	Bitmap - Open				
Name	PHBN 1		PASTE PHONE	Bitmap - Paste				
Owner	#IIIVEW01		JAIN ~	Bitmap - Pin				

12. Compile and run the form, which should look like the following:



13. Close the form. Leave the form **iiiBasicWinCtrls** open in the editor.

Check Box

A check box is used to represent a Yes/No choice.

There are three button states: checked, unchecked and grayed. Unchecked is the default. You would normally use the grayed state to indicate that the selections to which this check box applies have conflicting settings.

To gray out the check box, use the enabled property of the check box

```
#CKBX_1.Enabled := FALSE
```

You can use check boxes in groups to display multiple choices, from which the user can select one or more.

The ButtonState property indicates whether a check box has been checked The check box can be set to checked state programmatically, as shown in the following code:

#CKBX_1.buttonstate := checked

- 1. Select the *Design* tab for form iiiBasicWinCtrls.
- 2. From the *Controls* tab, drag and drop a *Check Box* onto the form.
- 3. Change the *Caption* property to **Member**.

Your form should look like the following:



4. Change the push button click event to test the *Check Box* ButtonState property and display a suitable message in a message box.

Your code should look like the following:

Evtroutine Handling(#PHBN_1.Click) If Cond(#CKBX_1.BUTTONSTATE *EQ CHECKED) Use Builtin(MESSAGE_BOX_SHOW) With_Args(*Default *Default *Defaul Else Use Builtin(MESSAGE_BOX_SHOW) With_Args(*Default *Default *Defaul Endif Endroutine

- 5. Compile and run the form. Ensure that the correct message box is shown, depending on the Check Box setting.
- 6. Close the form. Leave form **iiiBasicWinCtrls** open in the editor.

Edit Box

An edit box is an area where the user can enter text or where text can be displayed. The text can be set or retrieved using the Value property. An edit box is the default visualization style for fields.

- 1. Select the *Design* tab for the form **iiiBasicWinCtrls**.
- 2. Select the *Controls* tab, and drag and drop an *Edit box* onto the form. Note that the *Edit box* will be named EDIT_1.

- 3. Add a Push Button to the form below the Edit box. Note that your button will be named PHBN_2.
- 4. Create a Click event for PHBN_2. Change its *Caption* to **OK**.
- 5. Add logic to the new push button's *Click* event to change the value of the Edit box to "Hello".

Your code should look like the following:

```
Evtroutine Handling(#PHBN_2.Click)
#EDIT_1.value := 'Hello'
Endroutine
```

6. Compile the form and execute it to test the new push button and edit box. Your form should look like the following:

Basic windows Controls	
Test Member	hello

7. Close the form. Leave form **iiiBasicWinCtrls** open in the editor.

Group Box

A *Group box* provides a way of grouping components.

A *Group box* can optionally have a *Title*. Define a Title, by defining the *Caption* property of the *Group box*.

•	Options-	 • •	•••	• • •	

Typically, *Group boxes* are used to group together a set of *Check boxes* or *Radio buttons*, or a related group of fields.

·• Options	••••
: Home	
	•
: Office	
·•••••••••••••••••••••••••••••••••••••	

Group box has an EnableChildren property. When this property is set to **True**, all the components contained in this component are Enabled or Disabled, depending on whether the Enabled property of this component is set to **True** or **False**.

A *Group Box* will be used in the next part of this exercise.

Radio Button

Radio buttons are used to represent mutually exclusive choices. The convention is to enclose the radio buttons in a *Group box*.



The ButtonChecked property of the radio button indicates whether it is checked. By default the buttons are not checked.

While radio buttons and check boxes may appear to function similarly, there is an important difference: When a radio button is selected, the other radio buttons in the same group are automatically deselected. By contrast, any number of check boxes can be selected.

Radio Buttons are automatically "grouped" by their container, such as a Group Box, or a Panel.

If the radio buttons are simply placed on the form, then the form is their container.

- 1. Select the *Design* tab for the form iiiBasicWinCtrls.
- 2. Drag and drop a *Group Box* onto the form. Change the *Caption* property of the group box, to **Options**.
- 3. Adjust the size of the group box, so that two radio buttons can be added to it.
- 4. Drag and drop two radio buttons into the group box.

- a. Change their *Captions* to Home and Office respectively.
- b. Change the Home radio button *ButtonChecked* property to **True**.
- c. Adjust the width and height of each radio button if necessary.
- 5. Add a push button to the form, below the group box. Change its *Caption* to **Check**.
- 6. Create a *Click* event for the new push button.
- 7. Add logic to the new push button's click event to indicate which radio button is checked.

Your code should look like the following:

EVTROUTINE HANDLING(#PHBN_3.Click) IF COND('#RDBN_1.BUTTONCHECKED *EQ TRUE ') USE BUILTIN(MESSAGE_BOX_SHOW) WITH_ARGS(OK OK INFORMA ELSE USE BUILTIN(MESSAGE_BOX_SHOW) WITH_ARGS(OK OK INFORMA ENDIF ENDROUTINE

- 8. Compile and execute the form.
- 9. Ensure that the correct message is displayed in a message box, which indicates which radio button is checked.



10. Close the form. Leave iiiBasicWinCtrls open in the editor.

Label

Use a label to add text that the user cannot change.



Labels are usually used for instructions together with an Edit box. You enter the text for the label in its Caption property.

A Label has a click event, initialize event, double click event and so on.

Panel

A Panel provides a visual grouping of items in a similar way to a group box, but without a Title. A panel has properties to control the scroll bar such as *Horizontal Scroll, Vertical Scroll. HorizontalscrollPos, VerticalscrollPos* and so on.



A Panel has an EnableChildren property. All the components contained in this component are Enabled or Disabled when the Enabled property of this component is set to **True** or **False**.

Like the Group Box, the Panel is a container. Radio Buttons could be grouped on a panel.

Static Component

Use the Static component to draw lines, rectangles, triangles and ellipses on forms. The type of the image drawn is specified using the DrawType property.



- 1. Select the Design tab for form iiiBasicWinCtrls.
- 2. Drag and drop a Static component onto the form.
- 3. Change the *DrawType* property to **FrameIndentedEdge**. The Static component should look like the following:

·	1.	•		•	•	•	•	•	-	•	•	•	•	•	•	•	•	•		•	·
·	• Tr		_	_	_	_	-	-	-	-	-	-	_	-	-	_	_	-	Т	•	•
·	·																			•	•
٠																			h	•	•
·	· TI																		ΙT	•	•
٠	·																			•	•
·		_		_	_		_	_		_	_	_		_	_	_	_	_		•	•
·		•	• •	• •	•	•	•	•	1	•	•	•	•	•	•	•	•	•	÷	•	•

4. Experiment by changing the *DrawType* property to different values.

Status Bar

A Status bar creates a separate area at the bottom of the window where messages are displayed.

The status area can also contain other components, such as progress bars, (animated) pictures and edit boxes.

If more than one information message is issued, the user can scroll through them



A Status bar will be used in a later step in this exercise.

Track Bar

A Track bar allows the user to select a value with a slider.

<u> </u>	_	•	•	·	•	• •	• •	•	•	•		•	•	·		•	•	•	• •	•	•	•	•
	ſ		I.	1	ł	i	4	į	1	1	i	į		1	÷	i,	ł	1	1	i	į	1	1
			-			-		F	-	_	-	-	-			-	_	_		-		-	1
	1							_				-		_								_	_

A Track bar is a panel containing a slider and, optionally, tick marks.

Track bars are useful when you want to select a discrete value or a set of consecutive values in a range.

The user can move the slider by dragging it, clicking the mouse to either side of the slider, or using the keyboard.

- 1. Select the *Design* tab for the form iiiBasicWinCtrls.
- 2. Select the *Controls* tab, and drag and drop a *Track Bar* onto the form.
- 3. Change the *MaximumValue* property of the Track bar to **5**.

Note:

The *MinimumValue* has a default value of 1, so that there are now 5 divisions.

The *TickValue* has a default property of 1, so that there is one tick of the

Track bar for every increment of the Track bar.

- 4. Add a push button to the form, next to the Track bar. Change its *Caption* to **Show**.
- 5. Add a *Click* event for the new push button.
- 6. In the *Click* event for the Show push button, check for the track bar value and display a message in a EVTROUTINE HANDLING(#PHBN_4.Click)message box.

Your code should look like the following:

```
CASE OF FIELD(#TKBR 1.VALUE)
WHEN VALUE_IS(= 1)
USE BUILTIN(MESSAGE BOX SHOW) WITH ARGS(*Default *Default *
selected in track bar is 1')
WHEN VALUE IS(=2)
USE BUILTIN(MESSAGE BOX SHOW) WITH ARGS(*Default *Default *
selected in track bar is 2')
WHEN VALUE IS(=3)
USE BUILTIN(MESSAGE BOX SHOW) WITH ARGS(*Default *Default *
selected in track bar is 3')
WHEN VALUE IS(= 4)
USE BUILTIN(MESSAGE BOX SHOW) WITH ARGS(*Default *Default *
selected in track bar is 4')
WHEN VALUE IS(=5)
USE BUILTIN(MESSAGE BOX SHOW) WITH ARGS(*Default *Default *
selected in track bar is 5')
ENDCASE
ENDROUTINE
```

- 7. Compile and run the form.
- 8. Move the Track bar to a value and click the *Show* button. A message box is shown, indicating the value of the Track bar.

🛕 Test	Member	
Options Home		TRACK BAR
Office		Value selected in track bar is 2
Check		ОК
Enter the Options		
		0
		Show

9. Close the form. Leave form iiiBasicWinCtrls open in the editor.

Progress Bar

A Progress bar is used to indicate the progress of a lengthy operation. The progress bar is gradually filled with chunks from left to right as the operation progresses.

Use the MinimumValue and MaximumValue properties to set the range for the duration of the entire process. The *Value* property indicates the current value of the progress bar.

- 1. Select the *Design* tab for form iiiBasicWinCtrls.
- 2. On the *Controls* tab, select *All Controls* and drag and drop a *Status bar* onto the form. Note that it will be automatically located at the bottom of the form.
- 3. Drag and drop a *Progress bar* onto the right hand side of the *Status bar*.
- 4. Drag the left hand "handle" of the Progress Bar to the left to expand it. Your design should look like the following:



5. Change the *MaximumValue* property of the Progress bar to **20,000.0** Note that the *MinimumValue* property defaults to **1**.

- 6. Drag and drop a push button onto the form and change its *Caption* to **Loop**.
- 7. Create a *Click* event for the Loop push button.
- 8. Add the following logic to the Loop push button's *Click* event

```
EVTROUTINE HANDLING(#PHBN_5.Click)
#STD_COUNT := 1
#PGBR_1.VALUE := 1
BEGIN_LOOP USING(#STD_COUNT) TO(20000)
#PGBR_1.VALUE := #STD_COUNT
END_LOOP
MESSAGE MSGTXT('Process completed successfully')
ENDROUTINE
```

Review the code you just added. The *Progress* value is reset to **1**. A loop from 1 to 20000, will update the Progress bar Value for each step and display a message in the Status bar when the loop completes.

9. Compile and run the form. Verify that it works correctly.

				1	-0	
	S	ìhow]			
Process completed successfully						11

10. Close the form.

Note: This example is a simple example which shows how to use *Progress bar* properties. In practical applications, to show the progress of a lengthy process, set the value of the Progress bar at different places in the logic to visually display the progress of the operation.

Tab Folders and Tab Sheets

Go to Step 3. Create Tab Folder Form to see how to use a tab folder.

Step 3. Create Tab Folder Form

Use a *Tab Folder* to create a folder with several sheets to provide an organized way of showing a large amount of information.

Objectives

To create the form shown below, containing a tab folder with three tab sheets.

- When an employee number is entered on the Employee tab sheet, the employee details such as Surname, Given Name and Salary are to be displayed on the Details tab sheet.
- When the Skills tab is selected, a list of employee skills is to be shown

Note: This is a very simple example application and is not typical of the way that tab sheets interaction would be designed.

Employee Number A0090			
Tab Folders			
Employee Details Skills Employee Sumame Employee Given Name(s) Employee Salary	BLOGGS FRED JOHN ALAN 20.045 91 Tab Folders		
	Employee Details Skills Skill Code Grade Obtained for Skill Comment on skills acquired Date Skill Acquired (DDMMYY)	MARKET2 P aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	

1. Create a new form:

Name: iiiTabFolder

Description: A Simple Tab Folder

- 2. Drag and drop a *Tab Folder* to the form, and resize it to occupy most of the form.
- 3. With the *Tab Folder* selected, drag and drop a *Tab Sheet* onto the Tab Folder.
- 4. Click on the area to the right of the tab sheet, to select the *Tab Folder*, and drop another *Tab Sheet* onto the folder.
- 5. Repeat step 4. So that the *Tab Folder* contains 3 *Tab Sheets*.
- 6. Select the first Tab Sheet by clicking on the tab and then clicking on the main tab sheet area.

If you hover over the tab sheet, the component name will be shown in a tooltip:



- a. Change the tab sheet *Caption* property to **Employee**.
- b. Select the second tab sheet and change its *Caption* property to **Details**.
- c. Select the third tab sheet and change its *Caption* property to **Skill**.
- 7. On the *Repository* tab, expand the *Files* group and expand the file definition, PSLMST.
- 8. Select the Employee tab sheet and drop the EMPNO field onto it.
- 9. Select the Details tab sheet and drop SURNAME, GIVENAME and SALARY onto it.
- 10. On the *Repository* tab, expand the file definition PSLSKL.
- 11. Select the Skill tab sheet and drop fields SKILCODE, GRADE, COMMENT and DATEACQ onto it.
- 12. Create an Opening event routine for *Details* (TBSH_2).

Add code to initialize the fields SURNAME. GIVENAME and SALARY and then Fetch them from the file PSLMST with the key EMPNO.

Your code should look like the following:

Evtroutine Handling(#TBSH_2.Opening) Options(*NOCLEARMESSAGES * #SURNAME #GIVENAME #SALARY := *default Fetch Fields(#SURNAME #GIVENAME #SALARY) From_File(pslmst) With Endroutine

13. Create an Opening event routine for *Skill* (TBSH_3).

Add code to initialize the fields SKILCODE, GRADE, COMMENT and DATEACQ and then fetch them from the file PSLSKL with the key EMPNO.

Your code should look like the following:

Evtroutine Handling(#TBSH_3.Opening) Options(*NOCLEARMESSAGES * #SKILCODE #GRADE #COMMENT #DATEACQ := *default Fetch Fields(#SKILCODE #GRADE #COMMENT #DATEACQ) From_File([Endroutine

Note: The editor will give a warning because the key does not match the file keys. This logic will fetch the first skill record for the employee only. The actual file key is EMPNO and SKILCODE because the employee may have a number of skills.

14. Compile and run the form.

- a. On the Employee tab sheet enter and employee number such as A1012 or, A1013.
- b. Select the Details and then Skill to display employee data.

Summary

What I Should Know

- How to use the basic Visual LANSA Windows controls.
- How to use a simple Tab Folder

VFW010 – A Tab Folder Framework

Tab Folders

The Tab folder control has advanced docking, undocking and autohide features that can be used to transform a single Tab folder into a Tab framework application with virtual Tab folders attached to various parts of the window.

The virtual Tab folders can be automatically hidden (auto hide) and Tab sheets can be moved and attached to other parts of the screen (undock and dock).

The Tab folder properties manage the appearance of the framework and the individual Tab sheets manage the position and layout of a page in the tab framework.

The following Tab sheet properties are used:

DockPosition

Use the DockPosition property to specify where the Tab sheet is attached. The Tab sheet can be docked to the Center, Left, Right, Bottom or Top of the screen. Using the *DockPosition* properties of Tab sheets you can construct a tab framework application from a single Tab folder.

DockAllowUndock

Use the *DockAllowUndock* property to specify if the Tab sheet can be moved away (undocked) from the Tab folder. This property can be set to **True** or **False**. When this property is True, a docking bar (a double line) is displayed on the Tab sheet. To undock the Tab sheet, the user drags it by the docking bar.

DockCloseButton

Use the DockCloseButton property to specify whether a Close button is displayed in the Tab sheet. When a Tab sheet has a Close button, it can be closed. This property can be set to **True** or **False**.

DockAllowPositions

Use the *DockAllowedPositions* property to specify which part of the screen the Tab sheet can be attached to (docked). The values are Right, Top, Bottom and None. All positions can be selected. None overrides the multi-select options and specifies that the TabSheet cannot be docked anywhere. Note that this property applies to user interaction only and has no bearing on programmatic changes. Therefore a TabSheet with DockPosition(Left) and DockAllowedPositions(None) will appear as a DockLeft TabSheet.

Objectives

- The previous exercise demonstrated only the basic features of tab folders and tab sheets.
- This exercise uses some of the more advanced tab folder behaviour.

To achieve these objectives, you will complete the following:

Step 1. Create a FormStep 2. Define a Tab Folder FrameworkStep 3. Compile and Execute the FormSummary

Before You Begin

Complete the Simple Tab Folder exercise in VFW005.

Step 1. Create a Form

1. Create a form as shown in Step 1. Create a Form in exercise VFW005.

Name iiiTabFldFrm

Description A Tab Folder Framework



- 2. Enlarge the form to approximately *Width* 880 and *Height* 550.
- 3. Drag a *Tab Folder* onto the form and resize and position it to occupy all of the form.

You will learn more sophisticated form layout techniques in later exercises.

Step 2. Define a Tab Folder Framework

1. Right click on the *Tab Folder* (that is the background space next to the *Page 1* tab) and use the *Add Page* menu option to add 5 pages.

To add each page, you must first select the *Tab Folder*.

You can also right click on any tab sheet's tab, to select the tab folder and *Add Page*.



Your design should now look like the following:



2. Click on the Tab sheet Page 1 tab and then click in the main area to select it, and change the following properties:

Property	Value
DockAllowedPosition	Left+Right
DockAllowUndock	True
DockCloseButton	True
DockPosition	Left

Note: To see these properties, you must set up the editor to *Show Advanced Features*. See *File / Options / General*.

Your tab folder should now look like the following:



Hint: In the dropdown for *DockAllowPositions*, select the required values by using the Left Mouse button plus the Ctrl key.

3. Change the properties of the other *Tab sheets* as follows:

Tab Sheet 2

Property	Value
DockAllowedPositions	Left+Right
DockAllowUndock	False
DockCloseButton	False
DockPosition	Left

Tab Sheet 3

Property	Value
DockAllowedPositions	Left+Top
DockAllowUndock	True
DockCloseButton	True
DockPosition	Center

Tab Sheet 4

Property	Value
DockAllowedPositions	Left+Bottom+Right
DockAllowUndock	True
DockCloseButton	True
DockPosition	Bottom

Tab Sheet 5

Property	Value
DockAllowedPositions	Left+Right+Bottom
DockAllowUndock	True
DockCloseButton	True
DockPosition	Bottom

Your Tab folder should now look like the following:

Page2 Page1	Page3
	Page4 Page5

4. Save your form.

Step 3. Compile and Execute the Form

- 1. Compile and execute the form.
- 2. The Tab sheets can be shown as a separate window. Use the docking bar to drag the tab sheets.



3. Use the Pin to hide a Tab sheet.

	Tab Fol	der Fram	ework	_	
	Page1 F	age2	Page3		4
-		Ð	<u>×</u>		
han	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~				^
E A 1	ab Folde	er Framev	vork	_	_
A 1	Page 3	er Framev	vork		-
age1	Page3	er Framev	vork		
e2 Page1	Page3	er Framev	vork		
Page1	Folde Page3	er Framev	vork		
Page2 Page1	Page3	er Framev	vork		
Page2 Page1	Page3	er Framev	vork		

- 4. Click on a closed tab and click the Pin symbol to lock the Tab sheet in the open position.
- 5. Click on the Close (x) button to close a Tab Sheet.
Summary

Important Information

• The Visual LANSA Framework uses Tab folder features extensively to enable the interface to be tailored.

Tips & Techniques

- A Pop-up menu would be used to enable a closed tab sheet to be re-opened.
- Each Tab sheet tab may have an image assigned to it.

What I Should Know

• How to use the Docking, Undocking and Autohide features of the Tab folder.

VFW020 – Execute a Visual LANSA Framework Application

What is the Visual LANSA Framework (VL Framework)?

The VL Framework is an optional extension to Visual LANSA which provides an application Framework for designers and developers.

For the **end user**, the VL Framework provides consistent and well designed interface with many standard features which provide a powerful and flexible user interface. It is loosely based on Microsoft's Outlook interface design, which will be familiar to many end users.

For the **designer**, the VL Framework provides a rapid prototyping tool. The prototype will be developed into a real application.

For the **developer**, the VL Framework provides the tools to rapidly build a sophisticated modern application for Windows or Web deployment. It is especially beneficial if the developer is new to Windows or Web development. However, the power and flexibility of the Framework lends itself to most business applications and should be considered for all your LANSA projects.

VL Framework applications can be developed as Windows Rich clients, Web Browser based applications and Microsoft .Net applications.

If you are new to the VL Framework, you should study the *Visual LANSA Framework Guide*, the tutorials that it contains and the extensive shipped demonstration applications, which include all source code with extensive comments to enable their re-use.

These exercises provide an introduction to developing VL Framework applications for Windows, but time constraints mean that there is much more to learn in order to fully exploit the power and flexibility of VL Framework.

Objectives

- To execute a finished application in the VL Framework.
- To become familiar with the look and feel of VL Framework-based applications.
- To introduce some key concepts used when building VL Framework applications.

To achieve these objectives, you will complete the following:

Step 1. Execute the Visual LANSA Framework

Step 2. Execute an Application

Step 3. Using a Filter to Find an EmployeeStep 4. Using Commands and Command HandlersSummary

Before You Begin

• Check that you have met the Skills required as listed in *Other Requirements* in the *Visual LANSA Framework Guide*.

Step 1. Execute the Visual LANSA Framework

1. In Visual LANSA, use the *Tools* ribbon to select the *VL Framework* button to select the *Use Framework as User* option.



- 2. If you are opening a VL Framework for the first time, the latest demonstration version will opened by default. If you have only one framework the *Select Framework* dialog will not be shown.
- 3. If the *Select Framework File* dialog is shown, select the checkbox to *Open Latest Demonstration Version*.



The first time a framework is opened, the *Help Assistant* dialog is shown:

Help Assistant and Tutorial	
Visual LANSA Framework Tutor and Assistant	
Advanced Software Made Simple	
Start the Peip Assistant Start the Tutorial Don't Ask for These Details Again	

4. If the *Help Assistant* dialog is shown, unselect the *Start*.... options and select the checkbox *Don't Ask for These Details Again* and the click *OK*.

The framework uses XML to store the definition of your applications. The file vf_sy001_system.xml contains the latest demonstration system.

The Framework window will be displayed.



Step 2. Execute an Application

In this step, you will execute a shipped sample application. You will be introduced to *Business Objects, Filters, Instance Lists, Commands* and *Command Handlers*.

1. The *Navigation panel* on the left hand side, displays applications in a tree view.

As you click the different applications to expand them, you can see the business objects associated with them.

- 2. Select the *Programming Techniques* application. Then select the *Basic* application view.
- 3. Select the *Essential Business Object*.



Two new panels will appear. The top left panel is the filter, which is used to search employee data.

The right panel will show an instance list, containing the results of an employee search. This may contain saved values for the last search made.

	x			x
Specify a full or partial employee name.		Number	Name	
Employee Surname				
Clear the current list of Search				Clear List

Step 3. Using a Filter to Find an Employee

After a user has selected a business object, they typically want to locate a specific employee or list of employees. Filters allow you to search and sort the items for a business object.

1. Enter the letter B in the Employee Surname field and click the *Search* button. The instance list displays all employees whose surname begins with B.



Step 4. Using Commands and Command Handlers

In this step you will select an employee and review the Commands or actions which can be performed for an employee.

1. In the instance list, select the employee Veronica Brown. When an employee has been selected, the Basic details of the employee will appear in the bottom panel.

The essential pusine	rolect. Details (Abbro-Venturies billowing	
imployee Number	A0070	Save
mployee Surname	BROWN	
mployee Given Name(s)	VERONICA	
treet No and Name	12 Railway Street	
uburb or Town	Baulkham Hills	
tate and Country	NSW Australia	
ost / Zip Code	2220	
ome Phone Number	(02) 9609 4627	

By default, the Details command has been executed. The Details command handler displays the employee details.

2. Select the *File* menu and choose the *Exit* option to close the Visual LANSA Framework application.

Summary

Important Observations

- In Windows, the Visual LANSA Framework is executed as a Visual LANSA form.
- The Framework provides a consistent application interface. It is very easy to use, flexible and can be customized by the end user.

Tips & Techniques

- The end-user has the ability to fully customize the appearance of the application within the Framework. For example, panels can be positioned within the Framework or can be floated as separate windows. These capabilities are part of the Framework and are not coded by the developer.
- The Framework enables the end user to perform actions in many different ways.
- Commands can be executed using menus, toolbar icons and pop-up menus.

What You Should Know

- How to execute the Framework as an end user.
- How to execute an application created in the Framework.
- What are some of the features supported by the Framework.
- What are applications, business objects, filters, instance lists, commands and command handlers.

VFW030 – Create a Prototype

Objectives

- To create a prototype using the *Instant Prototyping Assistant*.
- To learn how to refine the prototype.



To achieve this objective you will complete the following:

- Step 1. Understand the Requirements
- Step 2. Create a Prototype iii HR Application
- Step 3. Define Filters and Command Handlers for Employees
- Step 4. Refine the Reports Business Object

Summary

Step 1. Understand the Requirements

You will define a prototype for a simple Human Resources application, which will consist of:

- Two business objects, Employees and Reports.
- Employees listed in the instance list based on searches by Name or by Location.
- Employees with actions, Details, New, Brief Notes, Image, Notes, Skills, Skills 2
- Reports with an action of Weekly, Monthly, Salary, Employee Query and Sort.

You will add further business objects in a later exercise.

Step 2. Create a Prototype iii HR Application

1. From the *Tools* ribbon, start VL Frameworks using the *VL Framework – as Designer* option.



2. If the *Select Framework File* dialog is shown, select the *Open Latest Demonstration Version* checkbox and click *OK*.

Alternatively, your trainer may inform you which framework name to use.

3. Once your Framework has loaded, start the *Instant Prototyping Assistant* from the *Framework* menu.

(Fr	amework) (Administration)	
	(New)	+
	(Properties)	
	(Applications)	•
	(Commands)	
	(Menus)	
	(Design Code Tables)	
	(Program Coding Assistant)	
	(Instant Prototyping Assistant)	
	(RAMP Tools)	
	(Virtual Clipboard)	Þ

4. Enter your new *Business Object* names, **Employees** and **Reports**, separated by a comma.

in Business Objects:	Employees, Reports	
estore previous values		
Employees Reports		

- 5. Click the *Next* button.
- 6. Actions will initially contain (the list at the bottom left) the defaults, *Details*, *New* and *Notes*. Define the additional actions required for Employees and Reports. These are, *Brief Notes*, *Image*, *Skills*, *Skills* 2, *Weekly*, *Monthly*, *Salary*, *Employee Query* and *Sort*. Separate each with a comma.

Actions: Details , N	ew , Notes , Brief Notes. Image, Skills, Skill	2, weekly, Monthly, Salary, Emploiyee Query,	Sort
Step 3. Drag and drop the a	ctions from the list below, onto vith many business objects.	all the appropriate business objec	cts in the list on the right
Details New Notes Brief Notes. Image Skills Skills 2 Weekly		Employees Reports	

7. Hold down the *Shift key* and select *Details, New, Notes, Brief Notes, Image, Skills and Skills 2* and drag them onto the *Employees* business object.

ctions: Detail	s , New , Notes , Brief Notes, Image, Ski	ls, Skills 2, weekly, Monthly, Salary, Employee (Query, Sort
Step 3. Drag and drop the The same action can be use Details New Notes Brief Notes Image Skils Skils 2 weekly Monthly Salary Employee Query	e actions from the list below, ed with many business objects.	onto all the appropriate business	objects in the list on the right Action: Details Employee Action: New Employee Action: Notes Employee Action: Brief Notes Employee Action: Image Employee Action: Skills Employee Action: Skills 2 Employee
Sort			<< Back Next >> Canc

8. Select the *Weekly, Monthly, Salary, Employee Query* and *Sort* actions and drag them onto the *Reports* business object.

Your business objects should now look like the following:



- 9. Click the *Next* button.
- 10. Using your initials instead of iii, enter a new application *iii HR Application*.
- 11. Drag the *Employees* and *Reports* business objects onto the *iii HR Application*.

Your iii HR Application should now look like the following:



- 12. Click the *Next* button.
- 13. On the final dialog, click the *Finish* button to generate your iii HR Application prototype.
- 14. From the Framework menu, Save and Restart your Framework.



Note: Your Framework definition is an XML file. It is good practice to regularly save your work. The Framework design tool will prompt you to save your Framework at regular intervals. (Ten minutes is the default setting). Each time you save a Framework, a copy of the Framework with the date and time appended is added to a \VF_Versions folder.

Step 3. Define Filters and Command Handlers for Employees

In this step you will define two filters for Employees and make a few basic enhancements to the prototype application.

1. Open the Business Object Properties dialog for Employees.

To do this, select the Employees business object on the *Navigation* panel and use the context menu (that is, the right mouse menu) to select the *Properties* dialog.



Alternatively, the *Properties* dialog will open if you double click on the business object in the *Navigation* panel.

The *Properties* dialog consists of a number of tab sheets which enable you to define and refine your application design. The Framework contains literally thousands of features which you enable or disable or refine, using one of these property tab sheets.

2. On the *Identification* tab, change the *User Object Name / Type* to **III_EMPLOYEES**. A business object named EMPLOYEES already exists because it is part of the shipped demonstration applications.

😹 Business Object Properties - Employees					
Identification Icons Visual St	yles Filters	Filter Settings	Commands Enabled	Command Display	Custom Properties
Caption	Employee	s		(ENG)	1
Caption (Singular)	Employee	8		(ENG)	
Hint:				(ENG)	
Sequence:	1				5
Internal Identifier:	7885CF25	5A6A3481B8A2EF	D874348C525		7
Unique Identifier:	106				(
User Object Name / Type		DYEES		Verify N	lame
Restricted Access	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~			~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

Note: The *Verify Name* button will check whether your *User Object Name* is unique within the framework. That is, not already used for an existing business object.

3. Select the *Icons* tab sheet and select any suitable icon for the Employees business object.



4. Select the *Filters* tab sheet. This tab enables you to define one or more filter which may be used with this business object. The *Filters* tab sheet itself contains three more tab sheets.

Business Object Properties - Employees			
Identification Icons Visual Styles	Filters Filter Settings Commands	Enabled Command Display Custom Properties SubT	ypes Instance List / Relat
. New Filter	Identification Icons Filter Snap-	-in Settings	
	Caption	New Filter	(ENG)
	Hint:		(ENG)
	Sequence:	2	
	Internal Identifier:	8A18A3737FA74967B5B5DCCB629328D7	
	User Object Name / Type	8A18A3737FA74967B5B5DCCB629328D7	Verify Name
	Last Changed	20121012-133119-JIVORY12	
	RAD-PAD File Name	C:\PROGRA~2\LANSA\X_WIN95\X_LANSA\x_trn\execu	te\RADPAD8A18

- a. The *Instant Prototyping Assistant* creates one filter for each business object. Change the *Caption* for the default filter (New Filter) to **By Name**.
- b. Select the *Icons* tab and select any suitable icon for the By Name filter.



- 5. Add a new filter by clicking the *New* button.
 - a. Change the new filter's *Caption* to **By Location**.
 - b. Select the *Icons* tab sheet and give the *By Location* filter any suitable icon.
- 6. You will now refine the *Command Handler* definitions for the Employees business object.

Select the *Commands Enabled* tab sheet. Note that its *Enabled* column contains the actions you defined in the *Instant Prototyping Assistant*.

a. Select the *New* command in the *Enabled* column.

Business Object Properties- Employees	
Identification Icons Visual Styles Filters Filter Settings	Commands Enabled Command Display Custom Properties SubTypes Instance List / Relations
To enable and disable commands drag them between these lists Not Enabled About About About Framework Al Details Address Al Details Address Al Entries Accounts Skills Skills Skills Skills 2	New (NEW) Choose Command Type Business Object Command Instance Command Sequence: 1 Command Options Stay Active Default Default Default Command No Show on Popup Menus
Assess Assistant Assistant Example 1 Assistant Example 2 Assistant Example 3 Attach Attach Backup Backup Backup	V Show on Instance List Tool Bar Hide All Other Command Tabs Alpha Arguments Restricted Access Alpha Argument 1: Execute as Hidden Command Numeric Argument 1: Bypass Locks Numeric Argument 2: Mumeric Argument 2: Numeric Argument 2: Menus Command Definitions
	Close

The information shown on the right hand side now contains the definition of the *New* command, as it will be used for the Employees business object.

Notice that in the *Choose Command Type* group box, the *Business Object Command* is selected. This is because the *New* command does not depend on selection from the instance list.

Using the *New* command would usually add a new entry to the instance list.

7. You will now specify how business object commands are displayed.

Select the *Command Display* tab sheet.

The property *Object Command Presentation* defines how business object commands are displayed.

a. Select Separate Stay on Top Window from the drop down.

Business Object Properties- Employe	es an	-0
Identification Icons Visual Styles Filter	s Filter Settings Commands Enabled	and Display Custom
Command Tab Style:	Tabs	-
Command Tab Show All:	Automatic	-
Command Tab Location:	Тор	•
Object Command Presentation	Separate stay on top window	$\overline{\mathbf{I}}$
Instance Command Presentation	Use part of the window	
Multiline Tab Sheet Captions		, i i i i i i i i i i i i i i i i i i i
Allow Float		······

- b. Click *Close* on the *Please Note* dialog which will appear. You will make further changes before you Save and Restart the Framework.
- 8. Click on the *Commands Enabled* tab sheet and click the *Command Definitions* button. Review the icons shown against each of the *Enabled* commands. You may find that some of your commands share the same icon. To change a command's icon, do the following:

Business Object Propertie	s- Employees Styles Filters Filter Setting a drag them between these lists Enabled Brief Notes Details Image New Notes Skills Skills 2	Commands Enabled Command Display Custom Properties SubTypes Instance List / Relations Choose Command Type Business Object Command Instance Command Sequence: 1 Command Options Stay Active Default Undows Undows Size Width Height Undows
Asprove Assess Assistant Assistant Example 1 Assistant Example 2 Assistant Example 3 Attach Attach Attach Attach Authorities Backup		Image: Show on Popup Menus -Optional Arguments Image: Show on Instance List Tool Bar -Optional Arguments Image: Hide All Other Command Tabs -Optional Argument 1: Image: Restricted Access Alpha Argument 2: Image: Execute as Hidden Command Numeric Argument 1: Image: Bypass Locks Numeric Argument 2:
Basic details Bookings Calculator Calendar Calendar Cancel Card Card Card		Command Handler Windows Component VF_CH001 Modk Up - RAD-PADRADPAD83AFBFE1C96C4F5DA75EFC6593524EBF.HTM Meruis Command Definitions.
		Close

a. Click the Command Definitions button (bottom right).

- b. Select the required command in the list on the left hand side.
- c. Select the *Bitmap and Icons* tab sheets and select a suitable icon. The change is saved automatically.
- d. Use the Scrollbar on the *Bitmaps* panel to select the same bitmap. This will ensure the instance list toolbar contains the same image as the command handler tab sheet.

Commands			
Commands	Sequence	-	Identification Toolbar and Menus Bitmaps and Icons other Options Usage
Bookings	1		-Icons - VF_IC389
Brief Notes	1		
Calculator	1		📡 🕺 🧕 🖉 🗐
🛗 Calendar	1	-	(g) 🗟 🖉 👘
Cancel	1		
Card Card	1		
S Category	1		
3 Charges	1		
Checks	1		
Cheques	1		
Claims	1		🚔 💀 👼 🛃
🙆 Close	1		
Command P	1		-Bitmaps (use Scrolbar to see more bitmaps) - VF_BM389
🔁 Connect	1		
🕄 Contacts	1		
Contents	1		🕒 🖾 (leg) 한 한 18 📧 🐮 👻 🖻 🖻 🖻 🖻 💌 🕷 📗
Copy	1		
Store Costs	1		
CRUD	1		
Custom Prop	1	-	
Delete	New		
			Close

Note: Do not spend time changing Icons and Bitmaps for all your Employee Commands. These could be changed at any time during development.

Hint: Clicking on the slider and using the cursor keys to scroll through the bitmaps provides more control than dragging the mouse.

- 9. Close the *Business Object properties* dialog. *Save and Restart* the Framework.
- 10. When the Framework has re-started, select your iii HR Application, followed by the Employees business object.
- 11. Confirm that Employees now has two filters.
 - a. Click the *Emulate Search* button and select an Employee.
 - b. Notice that the toolbar above the instance list contains your command handler icons.
 - c. Notice that all your instance list command handlers have a tab sheet at the bottom of the Framework, with the appropriate icon.

د			x				
Ry Name 🛗 By Location	🛛 💀 🖸 🔂	S 💈 🤠					
Filter for Employees.	Employee	Description					
This is a particular of a filter property used to achildren and	EMPLOYEE0001	Employee number 1					
Employees to be displayed.	EMPLOYEE0002	Employee number 2					
	EMPLOYEE0003	Employee number 3					
The user would normally enter search values here.	EMPLOYEE0004	Employee number 4	=				
	EMPLOYEE0005	Employee number 5					
To see what a filter does, click on the "Emulate Search"	EMPLOYEE0006	Employee number 6					
button.	EMPLOYEE0007	Employee number 7					
	EMPLOYEE0008	Employee number 8					
Program Coding Assistant Images Palette Emulate Search	EMPLOYEE0009	Employee number 9	-				
		Englance englas 10					
			x				
Employee : Details (EMPLOYEE0002-Employee number 2)							
Details Brief Notes 5 Image Notes 5 Skills 2	Skills						
This papel will bandle the action (or command) samed Detail	e for the husiness shi	at asmed Employees					
This panel will handle the action (or command) harried Detail	s for the business obj	ect named Employees.					
At the moment this panel is a prototype. When you have validated your prototype you would replace this panel with a real program.							
This panel is input capable. You may erase this text and add your own notes (and even pictures or images from the images palette).							
Any notes you add here are saved and may be used to help you flesh out your prototypes design and/or describe its functioning to others.							

- 12. Click in one of your Employee filter panels. Notice that these can be edited. The prototype panel is an HTML line editor. When adding text or images, use the *Enter* key to move to a new line.
 - a. Delete the existing text, add suitable text.
 - b. Click on the *Images Palette* button to display a dialog which enables images to be drag and dropped onto the filter panel.
 - Your objective is to make each panel "realistic" so that the prototype can be reviewed with and end user.
 - Restrict your changes to one panel. In a real project you would enhance the appearance of every filter and command handler panel.
- 13. Save and Restart the Framework.
- 14. Open the *Employees business object properties* dialog. Select the *Instance List / Relations* tab sheet.

A grid defines how the instance list columns are used and displayed. The third column *Caption* defines the column heading for columns shown in the instance list.

15. Change the *Captions* to **Number** and **Full Name**.

Business	Object Prop	erties - Employees				0		
Identification	n Icons Vi	sual Styles Filters Filter S	ettings Com	mands Enabled Comm	nand Displa	y Custom Properties	SubTypes Instance List / Rel	ations
Enable	Clear List But	ton						
Double	click for defai	uit command						
Save a	nd Restore In	stance Lists						
Allow m	nultiple selection	ons						
-								
V Allow	Instance List	to be sent to MS-Excel	File Prefix	to be used for MS-Exce	el	Spreadsheet_	(ENG)	
<u> </u>								
instance Lis	at Tool Bar Loc	ation	Тор			Ψ.		
Instance Lis	t Tool Bar Tex	t Location	<none></none>					
Instance Lis	t Tool Bar Hei	aht or Width	24					
			27					
Snap in Inst	tance List Brow	vser						
Sequence	Type	Caption	1.	Width % (Total 25%)	Decimals	Edit Code	Date/Time Output Format	UTC Conversion
10	VISUALID1	Number	25	5		Default	SYSFMT8	Local -> Local
20	VISUALID2	Full Name				Default	SYSFMT8	Local -> Local
	ACOLUMN1					Default	SYSFMT8	Local -> Local
	ACOLUMN2					Default	SYSFMT8	Local -> Local
	ACOLUMN3					Default	SYSFMT8	Local -> Local
	ACOLUMN4					Default	SYSFMT8	Local -> Local
3	ACOLUMN5				-	Default	SYSFMT8	Local -> Local
	ACOLUMN6					Default	SYSEMT8	Local -> Local

Note: The columns to be displayed are defined by giving them a Sequence number.

In your own application you would probably enable additional columns. Your filters must then be changed to populate the additional columns.

16. Close the *Business Object properties* dialog.

Step 4. Refine the Reports Business Object

- 1. Open the properties dialog for the *Reports* business object.
- 2. Select the *Filters* tab and delete the default "New Filter" which the *Instance Prototyping Assistant* has created. The command handlers for the Reports business object will not require an instance list.
- Select the *Commands Enabled* tab. Select each *Enabled* action and change its command type to *Business Object Command*. After the first change, the *Please Note* dialog will be displayed, deselect the *Warn me...* check box and click the *Close* button. When have changed all the commands to a Business Object Command, close the *Reports business object properties* dialog.
- 4. Now *Save and Restart* the Framework
- 5. Select the *Reports* business object for your application. The default *Weekly* command tab should be displayed. Right click on *Weekly* tab to show the context menu for the other Reports actions.
- 6. Save and Exit your Framework.
- 7. Restart the Framework as an end user and ensure it meets all the requirements.

Summary

Important Observations

- Due to time limitations, the prototyping section of these exercises is brief. When creating your own application prototype, this is a crucial step which may well determine whether your project gains acceptance or not.
- A well thought out prototype will clearly communicate what the finished application will look like and what it will do.
- A prototype should always be reviewed with the end user(s) in detail and signed off. With Visual LANSA Frameworks the prototype is completely re-used in the final application.
- To create a new application prototype with the Visual LANSA Framework, you simply set the application properties. You do not have to write any code.
- You can create application objects manually and you can create or extend an application using the Instant Prototyping Assistant.
- Applications can contain many business objects. A business object is the thing an end user works with, within an application.
- Filters enable end users to search for business objects.
- Command Handlers enable the end user to carry out business processes on business objects.
- A business object may have many command handlers.

Tips & Techniques

- Enhance the appearance of your prototype filters and command handlers using the images palette.
- The business object properties dialog enables the developer to refine the definition of the application, business object and its filters and command handlers.

What You Should Know

- How to create a prototype using the Instant Prototyping Assistant.
- How to refine the application design using the business object properties dialog.
- How to tune the behavior of command handlers.
- How to refine the appearance of prototype filters and command handlers, using text and images.

VFW040 – Snap in Real Filters

Objective

• To replace the prototype filters with real filters which will perform selection of employees for the instance list.

👶 Employees	COMPERSION			
<u>File Edit View Actions Help</u>	Windows (Framework) (Adminis	stration)	Employees By Name	
🕀 🍺 🤁 🕻	🧔 🧕 🙆	🗟 🧔	Filter	Quick Find
New Reports Transfer A	bout Web Site Resources Address	Charges Organizatio	on sookings spool Hies dow Queues	
Employees				
x		x		x
E 😭 Favorites	Sy Name 😥 By Location		🐻 🗆 🔟 🕹 😂 😒 🖕 👙 🛗	
HR Demo Application	7 Clear List	Search	Employee Description	Departm Salary Sta
Departments	V Credi Lot	Jearon	A1002 SMYTHE JOHN	ADM 25.000 01/
Employees	Employee Surname	_	A 1003 SMITHE Robert	FLT 31,000 21/
Q Reports	SM	_	A 1004 SMITHSON PAUL	AUD 21,000 01/
🗄 👸 iii Transport Application			A 1005 SMITHS PETER	ADM 46,700 01/
🗉 🞒 JI HR Application			A 1006 SMITHERS JACK	TRVL 25,000 01/
Marcus Instance Example				
Programming Techniques				
Administration				
	۰	F.	٠	•
	L		1	

To achieve this objective, you will complete the following:

- Step 1. Create Your Real By Name Filter
- Step 2. Snap in the Employees By Name filter
- Step 3. Review Filter Code

Step 4. Create a Real Employees By Location filter

Summary

Before You Begin

- Complete exercises VFW020 and VFW030.
- You may wish to review in the Visual LANSA Frameworks Guide: Filter in Key Concepts

Framework Programming

Step 1. Create Your Real By Name Filter

In this step you will create a real filter (a Reusable Part) which search the file PSLMST by surname, and populate the Instance List. You will learn how to use the *Program Coding Assistant*.

1. In the Framework, select the Employees business object and click the *Program Coding Assistant* button on the By Name filter.



You could also access the *Program Coding Assistant* from the *Framework* menu.

The *Program Coding Assistant* allows you to create different types of component which can be plugged into your filters, instance lists and command handlers.

Initially you will probably select filters which generate a complete component (for example, *Filter which searches by all logical views of a file*). As you become more expert, you might use a skeleton filter, or copy from one which is similar to what you need and complete the code manually.

2. If you are using a non-English system, click on *Framework-><Your Framework Description>* which is the first entry in the Coding Assistant's tree view. In the *Select the type of code you want to generate* panel, select *Set*

LANSA Code generation preferences. Click the *Next* button to set your coding preferences.

3. In the tree view select *iii HR Application / Employees / Filter->By Name*.



- 4. If your Framework is enabled for Web development, Select *Windows* as the target platform.
- 5. In the list headed *Select the type of code you want to generate*, select *Filter that searches using a file or view*.



- 6. Click the *Next* button. The coding assistant is a wizard which will present a series of panels which you will complete to generate the filter code required. There may be optional sections on these panels which you will complete as necessary to generate the logic required.
- 7. On the first panel enter PSLMST as the physical file name that most closely resembles the business object is.

Based on the file key and first two fields on the file, the coding assistant suggests the fields required to define the *Visual* and *Programmatic Identifiers*.

	Spe res bas	ecify the identifica embles this busine sic identification p physical file that r	ation protocol y ess object spe rotocol for you most closely	Filter that searches using a file or w you have decided to use for this busines cify its name and the assistant will attem u. PSLMST Personnel	iew is object. If a physical file ipt to automatically deduce a	-	
	VIS	UAL IDENTIFIERS	(for building V	isualID1 and VisualID2 values)			
11		Field Name	Туре	Description	Drop Selected		
Н	1	EMPNO	ALPHA	Employee Number			
Ш	2	SURNAME	ALPHA	Employee Surname	Drop All		
Ц	3	GIVENAME	ALPHA	Employee Given Name(s)			
1	4						
	5			-			
	Add fields from this Physical File Add Keys Add Al						
1	PRO	GRAMMATIC IDE	NTIFIERS (for	building AKey1,2,3,4,5 and NKey1,2,3,	4,5 values)		
		Field Name	Type	Description	Drop Selected		
N	1	EMPNO	ALPHA	Employee Number			
	2				Drop All		
	3			-			
	Add fields from this Physical File Add Keys Add All						

- A **Visual Identifier** is the field or fields that a user would use to identify a unique instance of an object
- A **Programmatic Identifier** is the field(s) that the program would use to identify a unique instance of the business object. Typically these would be the primary keys of the file or files that make up the data in the instance list.
- The additional columns should be completed, if necessary, to correspond to any additional columns which you have added to your instance list, during the prototyping phase.

In this case, the generated code will use EMPNO and FULLNAME (based on SURNAME plus GIVENAME) to define the visual identifiers.

EMPNO defines the programmatic identifier

- 8. Click the *Next* button.
- 9. On the next panel, select PSLMST2 as view to be used for filtering /searching operations.
- 10. Select SURNAME as the key(s) of the *selected view, to be used for search operations*. SURNAME will be the input field on the filter panel.

Select the following options:

• Allow generic searching

- Remember key values between filter executions
- Allow the user to clear the instance list

Specify the underlying physical file that will be searched by this filter	PSLMST	Personnel		
Select the view to be used for filtering / se operations:	arching	Select the key(s) search operations	of the select	ed view to be used for
View Name Description		Key Name	Type	Description
PSLMST Personnel		SURNAME	ALPHA	Employee Surname
PSLMST1 Personnel by Deptment, Se PSLMST2 Personnel by Surname, Giv	ven Name	GIVENAME	ALPHA	Employee Given N
User must specify all chosen keys				
Allow generic searching				
Remember key values between filter e	xecutions			
Allow user to dear instance list				

11. Click the *Next* button. Select the check box *Routine to listen for signals to update the instance list.*

Filter that searches using a file or view
Include uTerminate routine Deutine to lister for signals to undate the instance list
Check still connected before doing database IO

12. Click the *Generate Code* button.

The generated code will be displayed.

You now simply need to create the component by specifying its *Name and Description* and clicking the *Create* button. VLF code assistant, currently limits reusable part names to 9 characters.

13. Specify **iiiVFW04** as the filter name, where **iii** are your initials, and **Employee By Name Filter** as the description. If you are using an unlicenced or trial version of Visual LANSA your component names must be of the form DEMCOM05. Click the *Create* button.

When you click the *Create* button, a reusable part will be created in the Repository and the code will be copied into the editor.

After a brief delay, a message *Created in the development environment* will be displayed.

14. Switch to Visual LANSA and compile the reusable part.

Step 2. Snap in the Employees By Name filter

Now that you have created and compiled your filter, you need to snap it into the Framework. This means that you will replace a dummy filter (which is a shipped component) with your reusable part.

- 1. In the Framework, close the *Program Coding Assistant*.
- 2. Double click the Employees business object to display its *Business Object* properties dialog.
- 3. Select the *Filters* tab sheet and select the By Name filter. Then select the *Filter Snap-in Settings* tab sheet.
- 4. Specify iiiVFW04 as the *Windows Filter handler* component, where iii are your initials.

Business Object Properties - Employees							
Identification Icons Visual Styles	Filters Filter Settings Commands Enabled Command Display Custom Properties S						
🛃 By Name	Identification Icons Filter Snap-in Settings						
By Location	Stay Active Default Filter Handler Windows Component IIIVFW04						
	Mock Up - RAD-PADRADPAD8A18A3737FA74967B5B5DCCB629328D2						

- 5. Save and Restart your framework.
- 6. Select the Employee business object. On the By Name filter, enter a single letter such as "S" or "B" and click the *Search* button. The Instance List should be populated with matching employee details.

Sy By Name 🔯 By Location	x		
🔽 Clear List	Search	Employee	Description
Employee Surname		A0070	Brown Veronica
В		A0090	Bloggs Fred Blake John
		A1404	Black Gillian
		A3564	Brown Freddy

Step 3. Review Filter Code

Although you can create simple filters using the *Program Coding Assistant*, you should understand how they are coded.

- 1. Switch to the Visual LANSA editor, where the reusable part, iiiVFW04 is still open.
- 2. Use the *GoTo* tab to select the uSelectData method routine. Double click on a routine on the *Go To* tab and the editor positions to that line.

The filter notifies the Framework that an update is about to occur:

* Indicate that Employees instance list updating is about to start Invoke Method(#avListManager.BeginListUpdate)

The data is selected using a SELECT/ENDSELECT loop

* Select appropriate instances of Employees
Select Fields(#XG_Ident) From_File(PSLMST2) With_Key(#XG_Keys) Nbr_
* Set up the visual Identifier(s)
Change Field(#UF_VisID1) To(#EMPNO)
Change Field(#UF_VisID2) To(#SURNAME)
Use Builtin(BConcat) With_Args(#UF_VisID2 #GIVENAME) To_Get(#UF_V

The Visual ID fields are set up with EMPNO and SURNAME + GIVENAME.

Then the data is added to the instance list.

* Add instance details to the instance list Invoke Method(#avListManager.AddtoList) Visualid1(#UF_VisID1) Visualid2

Visualid1 will be shown in column one of the instance list and Visualid2 will be shown in column two. Akey1 is the key that uniquely identifies an employee. If it was a numeric value, NKey1 would be used.

Finally, the Framework is notified that the instance list update is complete:

•••

Endselect

* Indicate that Employees instance list updating is now complete Invoke Method(#avListManager.EndListUpdate)
Note: avListManager is a component which is part of the filter ancestor VF_AC007.

- 3. For this step you need to ensure that your editor settings are showing advanced features. Use the *Options* menu to display the *Settings* dialog.
- 4. Select the *General* settings and ensure that under *Details*, *Show Advanced Features* is selected.
- 5. Select the *Details* tab in the editor, to display your component properties. Note that the *Ancestor* property is **#VF_AC007**. All filters must inherit this base class, which provides a pre-defined set of events, properties and methods.
- 6. Select the *Outline* tab in the editor, to see the components you inherit from VF_AC007.
- 7. Use the context menu on the component avListManager and select the *Features* option.
- 8. Expand the methods and examine them. Double click on the AddtoList method and note that the Help tab contains help for this method.
- 9. Close the component iiiVFW04. You may want to read the *Windows Filter* and *Command Handler Anatomy* in the *Visual LANSA Framework Guide* to learn more about how these components are structured.

Step 4. Create a Real Employees By Location filter

The *By Location* filter could quickly be created using the *Program Coding Assistant*. However, in this step you will create a new filter component by creating much of the code yourself. This will enable you to gain a greater understanding of writing code for the Framework.

1. From the *File* menu click the *New* button select *Reusable Part* :



Then select **Panel**:



3. Create a new Reusable Part / Panel:

Name: iiiVFW05

Description: Employees by Location Filter

Enable for RDMLX = **Yes**

A reusable part is created with an ancestor of PRIM_PANL which means it is a visual component.

4. Select the *Details* tab and change the *Ancestor* to **VF_AC007**. This is the filter base class which gives your filter component the properties, events and methods it needs to interact with Framework components.

Your editor must be configured to *Show Advanced Features*. Use *File / Options* to change this setting under *General*, if necessary.

- Paste the source code from VFW040 Appendix replacing the existing code. A number of errors will be flagged. Ignore all of these, since you will now complete the code required.
- 6. Switch to the *Design* tab, and resize the panel so that it looks like this:



- 7. The next step requires the *Layout Helper* tab to be visible.
 - a. Select *Views* on the *Home* ribbon:



b. Select *Layout Helper*:



The *Layout Helper* will now be displayed on the left hand side.

8. In the *Layout* dropdown, select MAIN_LAYOUT.

Layout Helper	
Layout Managed Component	Ē
IIIVFW05	*
Layout	5 X
MAIN_LAYOUT	•
Children 🕧 🗙 🖉 🗙	Child Details As Child Details
BODY_PANEL	Category Processing Order
~~~~~	Vertical

Note that the BODY_PANEL and BUTTON_PANEL are Children of MAIN_LAYOUT.

The supplied code includes attachment managers and flow managers which define your filter as:

Reusable part iiiVFW05 (a panel) attachment manager (MAIN_LAYOUT)

BODY_PANELflow down manager (BODY_FLOW)

BUTTON_PANEL

flow down manager (BUTTON_FLOW)

- 9. On the *Repository* tab, expand *Files* and select file SECTAB. Drag and drop fields DEPTMENT and SECTION onto the main panel, below the Clear List checkbox. This is the BODY_PANEL.
- 10. Select each field and on the *Details* tab.
  - a. Change the *MarginLeft* property to **100**.
  - b. Reduce the width of each field, which are actually 4 and 3 characters long.

Your filter should look like the following:



**Hint:** For this type of change, you can select both fields (hold down the Shift Key) and then change *MarginLeft* to the required value.

- 11. Switch back to your source code.
  - a. Complete the Fields() parameter of the Group_by XG_KEYS. Replace < FILE KEYS > with fields DEPTMENT and SECTION.
  - b. Change the Cond() parameter in the Def_Cond statement. Replace <INPUT FIELD> with field DEPTMENT

Your changed code should look like the following:

Group_By Name(#XG_Keys) Fields(#DEPTMENT #SECTION)

•••

```
Def_Cond Name(*SearchOK) Cond(#deptment *ne *Blanks)
```

**Note:** The Group_by XG_IDENT is already correctly defined, to retrieve fields EMPNO, SURNAME and GIVENAME.

- 12. Complete the uSelectData method routine. The supplied source contains a number of comment lines which indicate where you need to add code.
  - a. Invoke the **BeginListUpdate** method in the List Manager

# av List manager. begin List Up date

b. If the Clear_List ButtonState is checked, invoke the ClearList method in the List Manager:

```
If ('#Clear_List.ButtonState = Checked')
#avlistmanager.ClearList
```

Endif

c. Select the Group_by XG_IDENT from logical file PSLMST, with keys in Group_by XG_KEYS, using Generic = *yes and Number of keys = *compute.

Select Fields(#XG_Ident) From_File(pslmst1) With_Key(#XG_Keys) Nbr_Ke

- d. Place the ENDSELECT before the * Indicate that employees instance list updating is complete.
- e. Set up work field uf_visid2 as "Surname, Givename"
  #uf_visid2 := #surname + ', ' + #givename

f. For each employee selected, invoke the AddtoList method in the List Manager with appropriate parameter values for Visualid1 and Visualid2 and Akey1.

#avlistmanager.addtoList Visualid1(#empno) Visualid2(#uf_visid2) Akey1(

g. Invoke the EndListUpdate method in the List Manager (after the EndSelect)

#avlistmanager.endListUpdate

Your completed code should now look like the following:

Mthroutine Name(uSelectData)

```
* Save the current key values from overwrites done by the select loop Inz_List Named(#Save_Keys)
```

* Indicate that Employees instance list updating is about to start #avlistmanager.beginListUpdate

```
* Clear the current Employees business object instance list
```

If ('#Clear_List.ButtonState = Checked')

#avlistmanager.clearList

Endif

```
* Select appropriate instances of Employees
```

Select Fields(#XG_Ident) From_File(pslmst1) With_Key(#XG_Keys) Nbr_Ke * Set up the visual Identifier(s)

```
#uf_visid2 := #surname + ' ' + #givename
```

```
* Add instance details to the instance list
```

#avlistmanager.AddtoList Visualid1(#empno) Visualid2(#uf_visid2) Akey1(#e
Endselect

```
* Indicate that Employees instance list updating is now complete #avlistmanager.EndListUpdate
```

* Restore the saved key values

```
Get_Entry Number(1) From_List(#Save_Keys)
```

Endroutine

# 13. To complete the event handling routine for DEPTMENT.Changed, replace <INPUT FIELD > with #Deptment in this line:

Evtroutine Handling(< INPUT FIELD >.Changed) Options(*NOCLEARMES!

- 14. Compile your component.
- 15. In your Framework, open the *Employees business object properties* dialog and select the *Filters* tab.
- 16. On the *Filters* tab, select the *Filter Snap-in Settings* tab and snap in the Windows component as iiiVFW05.

🔗 Business Object Properties - Er	nployees	
Identification Icons Visual Styles	Filters Filter Settings   Commands Enabled   Command Display   Custom Propert	ties
🛃 By Name	Identification Icons Filter Snap-in Settings	ļ
By Location	Stay Active Default 💌	1
	- Filter Handler	
	Windows	
	Component IIIVFW05	
	Mock Up - RAD-PADRADPAD3B4CF79C1D654FEDB2CAA2F9F31	F460
		1

- 17. *Save and Restart* the Framework and test the *By Location* filter.
- 18. Test the filter by specifying a partial department code only (for example, "A") and also with both a department and section specified (for example, ADM and 01).

This filter has a very basic user interface. In a later exercise you will learn how to enhance it.

You will learn more about using *Layout Managers* in a later exercise.

### Summary

### **Important Observations**

• With snap-in real filters you create real functionality in your application.

# **Tips & Techniques**

• The source code for filters used in the shipped demonstration applications can be found in components named DF_*.

# What You Should Know

- How to create filters.
- How to snap filters into the Framework.
- How to use the *Program Coding Assistant*.
- How to develop a filter by writing your own code.

# VFW040 – Appendix

* Simple Field, Group and Condition Definitions *	Source Code for Filter iiiVFW05 Function Options(*DIRECT) Begin_Com Role(*EXTENDS #VF_AC007) Height(170) Layoutmanager(#MAIN_LAYOUT) Width(345) *
Group_By Name(#XG_Keys) Fields(< FILE KEYS >) Group_By Name(#XG_Ident) Fields(#EMPNO #SURNAME #GIVENAME) Def_List Name(#Save_Keys) Fields(#XG_Keys) Type(*Working) Entrys(1) Def_Cond Name(*SearchOK) Cond(< INPUT FIELD > *ne *Blanks) * ====================================	* Simple Field, Group and Condition Definitions * ====================================
<ul> <li>* Component definitions</li> <li>* ====================================</li></ul>	Group_By Name(#XG_Keys) Fields(< FILE KEYS >) Group_By Name(#XG_Ident) Fields(#EMPNO #SURNAME #GIVENAME) Def_List Name(#Save_Keys) Fields(#XG_Keys) Type(*Working) Entrys(1) Def_Cond Name(*SearchOK) Cond(< INPUT FIELD > *ne *Blanks) * ====================================
<ul> <li>* Body and Button arrangement panels</li> <li>Define_Com Class(#PRIM_PANL) Name(#BODY_PANEL) Displayposition( Define_Com Class(#PRIM_PANL) Name(#BUTTON_PANEL) Displaypositic</li> <li>* Attachment and flow layout managers</li> <li>Define_Com Class(#PRIM_ATLM) Name(#MAIN_LAYOUT)</li> <li>Define_Com Class(#PRIM_ATLI) Name(#BODY_ATTACH) Attachment(Cer</li> <li>Define_Com Class(#PRIM_ATLI) Name(#BUTTON_ATTACH) Attachment(I</li> <li>Define_Com Class(#PRIM_FWLM) Name(#BUTTON_FLOW) Direction(Top</li> <li>Define_Com Class(#PRIM_FWLM) Name(#BUTTON_FLOW) Direction(Top</li> </ul>	* Component definitions *
<pre>Define_Com Class(#PRIM_PWLM) Name(#BOD Y_FLOW) Direction(TopTo Define_Com Class(#PRIM_PHBN) Name(#Search_Button) Buttondefault(Tru Define_Com Class(#PRIM_FWLI) Name(#FWLI_Search_Button) Manage(#S * Define the fields and components that are on the filter form Define_Com Class(#PRIM_CKBX) Name(#CLEAR_LIST) Caption('Clear Lix Define_Com Class(#PRIM_FWLI) Name(#FWLI_CLEAR_LIST) Manage(#C Define_Com Class(#PRIM_FWLI) Name(#FWLI_1) Parent(#BODY_FLOW) Define_Com Class(#PRIM_FWLI) Name(#FWLI_2) Parent(#BODY_FLOW) * ====================================</pre>	<ul> <li>* Body and Button arrangement panels</li> <li>Define_Com Class(#PRIM_PANL) Name(#BODY_PANEL) Displayposition(</li> <li>Define_Com Class(#PRIM_PANL) Name(#BUTTON_PANEL) Displaypositi</li> <li>* Attachment and flow layout managers</li> <li>Define_Com Class(#PRIM_ATLM) Name(#MAIN_LAYOUT)</li> <li>Define_Com Class(#PRIM_ATLI) Name(#BODY_ATTACH) Attachment(Ce</li> <li>Define_Com Class(#PRIM_ATLI) Name(#BUTTON_ATTACH) Attachment(Ce</li> <li>Define_Com Class(#PRIM_FWLM) Name(#BUTTON_FLOW) Direction(To</li> <li>Define_Com Class(#PRIM_FWLM) Name(#BUTTON_FLOW) Direction(To</li> <li>Define_Com Class(#PRIM_FWLM) Name(#BODY_FLOW) Direction(Top Te</li> <li>Define_Com Class(#PRIM_FWLI) Name(#FWLI_Search_Button) Manage(#</li> <li>* Define the fields and components that are on the filter form</li> <li>Define_Com Class(#PRIM_FWLI) Name(#FWLI_SEAR_LIST) Caption('Clear List Define_Com Class(#PRIM_FWLI) Name(#FWLI_CLEAR_LIST) Manage(#W</li> <li>Define_Com Class(#PRIM_FWLI) Name(#FWLI_1) Parent(#BODY_FLOW)</li> <li>Define_Com Class(#PRIM_FWLI) Name(#FWLI_2) Parent(#BODY_FLOW)</li> </ul>

Mthroutine Name(uInitialize) Options(*Redefine)

* Restore Clear List button state

Invoke Method(#avFrameworkManager.avRestoreValue) Withid1(*Componen

* Enable/disable the search button as appropriate

Set Com(#Search_Button) Enabled(*SearchOK)

Endroutine

Mthroutine Name(uSelectData)

* Save the current key values from overwrites done by the select loop Inz_List Named(#Save_Keys)

* Indicate that Employees instance list updating is about to start

* Clear the current Employees business object instance list

If ('#Clear_List.ButtonState = Checked')

Endif

```
* Select appropriate instances of Employees
```

* Set up the visual Identifier(s)

* Add instance details to the instance list

* Indicate that Employees instance list updating is now complete

* Restore the saved key values

```
Get_Entry Number(1) From_List(#Save_Keys)
```

*

Endroutine

Evtroutine Handling(< INPUT FIELD >.Changed) Options(*NOCLEARMES: Set Com(#Search_Button) Enabled(*SearchOK) Endroutine

* _____

* Handle the search button

Evtroutine Handling(#Search_Button.Click) Options(*NOCLEARMESSAGE

Invoke Method(#Com_Owner.uSelectData)

Endroutine

End_Com

# VFW042 – Snap in a Real Command Handler

## Objective

- To replace a prototype command handler with a real component that will perform the processing required.
- To replace the Details command handler with a real command handler.

🍰 Employee : Details (A1	DD2-SMYTHE JOHN)
Details Brief Notes	🔟 Images   💊 Notes   📇 Skills 2   👙 Skills
Employee Number	A1002 Save
Employee Surname	SMYTHE Back
Employee Given Name(s)	NHOL
Street No and Name	20 Cobbitty Avenue,
Suburb or Town	WERRINGTON.
State and Country	NSW.
Post / Zip Code	2100
Home Phone Number	047 629 0442
Business Phone Number	798 4381
Department Code	ADM
Section Code	01
Employee Salary	25,000.04
Start Date (DDMMYY)	1/01/77
Termination Date (DDMMYY)	0/00/00
1	Messages Ready Local ENG JIVORY12 7/08/12 13:53

To achieve this objective, you will complete the following:

Step 1. Create a Real Command Handler Step 2. Review Command Handler code Step 3. Snap in Real Command Handler Summary

# **Before You Begin**

Complete exercises VFW020, VFW030 and VFW040.

You may wish to review:

• *Commands* in *Key Concepts* in the *Visual LANSA Frameworks Guide*.

# **Step 1. Create a Real Command Handler**

In this step you will create a real command handler for the Details command.

- 1. From the *Framework* menu, start the *Program Coding Assistant*.
- 2. Select the *iii HR Application / Employees / Details* command handler.
- 3. If necessary select *Native MS Windows* as the platform.
- 4. Select *Basic Command Handler* as the type of code. This command handler, displays the fields you specify for the object selected in the instance list. It includes a *Save* button. You must complete the required update logic for this button.

The *CRUD Command Handler*, is used with a *CRUD filter*. The commands defined for the business object must be *New*, *Details*, *Copy* and *Delete*.

The *Command Handler that maintains a list*, generates a command handler for the business object data as well as providing a list of records from a related file.

- 5. Click the *Next* button.
- 6. Specify file name PSLMST as *The physical file which most closely resembles this business object.*

As before the code assistant will select the key(s) of the file and the first two fields from the file, which are suitable in this case.

- 7. Click the *Next* button.
- 8. Specify file name PSLMST in the *Add fields from this physical file* section.
- 9. Click the *Add All* button.

con tha	mand handler p t code assistant	banel. Your o	hoice to put fields onto the head your command execution	er and/o	r list areas will affect the	way
-	de thest was a second					
iek	Field Name	Type	Description	der 🔺	Drop Selected	
1	EMPNO	ALPHA	Employee Number		Drop Science	
2	SURNAME	ALPHA	Employee Surname		Drop All	
3	GIVENAME	ALPHA	Employee Given Name(s)			
4	ADDRESS1	ALPHA	Street No and Name			
5	ADDRESS2	ALPHA	Suburb or Town			
6	ADDDESS3	ALDHA	State and Country	-		
dd	fields from this	Physical File	PSLMST Add Key	6	Add All	
_				~	ferrenand	

- 10. Scroll down the list of fields. Select the following fields and use the *Drop Selected* button to remove them:
  - STARTDTER
  - TERMDATER
  - MNTHSAL
- 11. Click the *Next* button.
- 12. Select only the Include Default Save Button and Logic.



13. Click the *Generate Code* button.

The generated code is displayed.

- 14. Specify iiiVFW06 as the name of the component and *Employee Details Command Handler* as the description.
- 15. Click the *Create* button.

- 16. After a few moments the *Created in development environment* message will be displayed.
- 17. Switch to the Visual LANSA editor.
- 18. Use the *GoTo* tab to find the SAVE_BUTTON click event routine.

Add an UPDATE statement to update group_by fields XG_HEAD in the file PSLMST. Your code should look like the following:

Update Fields(#XG_HEAD) In_File(pslmst) Val_Error(*next)

19. Compile your component.

# Step 2. Review Command Handler code

1. Locate the uExecute method. This method is called by the Framework when an employee business object is selected in the instance list.

Note that this method routine redefines the uExecute method defined in the command handler ancestor, which is VF_AC010.

- 2. The uExecute method is invoked in the ancestor. This ensures that any standard logic in the ancestor's uExecute method in performed.
- 3. The List Manager's GetCurrentInstance method is invoked. In this case only AKey1 is requested, but this method could request any of the columns defined in the business object's instance list.

The uExecute method is then able to fetch the fields required from the employee record, using the key of EMPNO.

# Step 3. Snap in Real Command Handler

- 1. In the Framework, open the *Employees business object* properties dialog.
- 2. Select the *Commands Enabled* tab.
- 3. In the list of *Enabled* commands, select *Details*.
- 4. Specify iiiVFW06 as the command handler Windows component.

Business Object Properties - Employees	
Identification I cons Visual Styles Filters Filter Settings To enable and disable commands drag them between these lists  Not Enabled About About Framework Accounts Address Al Details Al Details Al Details Anount Assistant Example 1 Assistant Example 1 Assistant Example 2 Assistant Example 2 Attach Attachments Authorities Backup	Commands Enabled       Immand Display       Custom Properties       SubTypes       Instance List / Relations         Details (DETAILS)       Choose Command Type       Business Object Command       Instance Command         Sequence:       1       Command Options       Own Window Size         Stay Active       Default ×       Width Height         Version       Show on Popup Menus       Vindows         Show on Instance List Tool Bar       Optional Arguments         Hide All Other Command Tabs       Apha Argument 1:         Restricted Access       Numeric Argument 1:         Bypass Lookis       Numeric Argument 2:
Basic details Bookings Calculator Calendar Calendar Carcel Carcel Category Category	Component IIINLW05 Modk Up - RAD-PADRADPAD745AA 19AA 52D 43ED8E 1876F265DCB 159.HTM

- 5. Close the properties dialog.
- 6. Use a filter to populate the Employees instance list.
- 7. Select an employee in the instance list. Your command handler is snapped into the Framework and usable.
- 8. Make a change to the employee and save the changes.

Note that currently the instance list does not reflect changes to an employee (for example, a change of Surname). In a later exercise you will learn how to handle this situation.

## Summary

### **Important Observations**

• The Details command handler is a simple update program. In your own applications you would embed other business logic and processing into the command handler as necessary.

# **Tips & Techniques**

- For more information read the Windows Filter and Command Handler Programming in the Visual LANSA Frameworks Guide.
- The source code for the shipped demonstration applications can be found in components named DF_*.

# What You Should Know

- How to use the Program Coding Assistant to create your own command handlers.
- How to snap a command handler into the Framework.
- The VLF allows you to prototype and rapidly build and deploy an application with no OO knowledge.
- Creating this style of application enables you to build on your Visual LANSA knowledge, to rapidly build the style of application your end users will expect.

# VFW044 – Add Instance List Columns

# Objective

- To add columns to the instance list.
- To modify a filter to populate the additional instance list columns
- To add an alpha, a numeric and a date column to the instance list.



**Note:** In this exercise you will modify only the *By Name* filter. Normally you would make the same modifications to *By Location* filter and to any other filters which populate the Employees instance list.

To achieve these objectives you will complete the following:

Step 1. Add columns to the Instance List

Step 2. Change the Filter

Summary

# **Before You Begin**

- You should complete exercises VFW020, VFW030, VFW040 and VFW042.
- You may wish to review Adding Additional Columns to Instance Lists in the *Visual LANSA Frameworks Guide*.

# Step 1. Add columns to the Instance List

In this step you will configure your Employees business object to make more columns visible in the instance list.

- 1. Start the Framework as a Designer.
- 2. Open the properties dialog for the *Employees business object*.
- 3. Select the Instance List / Relations tab.

Two visual identifiers are already defined.

Business	Object Prop	erties - Employe	-5					
entification	Icons Vis	ual Styles   Filters	Filter Settings	Commands Enabled Com	mand Displa	y Custom Properties	SubType Instance List / Re	ations
Enable	Clear List Butt	on						
Double	click for defau	It command						
Save an	nd Restore Ind	stance Lists						
Allow m	ultiple selectio	ns						
Allow 1	Instance List t	o be sent to MS-E	xcel File Pr	efix to be used for MS-Ex	cel	Spreadsheet_	(ENG)	
stance List	t Tool Bar Loca	stion	Тор					
stance List	Tool Bar Tex	t Location	<none></none>	<none> *</none>				
stance List	t Tool Bar Heig	ht or Width	24					
nap in Insta	ance List Brow	ser						
Sequence	Type	G	aption	Width % (Total 25%)	Decimals	Edit Code	Date/Time Output Format	UTC
10	VISUALID1	Number		25		Default	SYSEMT8	Loca
10	VISUALID2	Full Name				Default	SYSFMT8	Local
	ACOLUMN1					Default	SYSFMT8	Loca
	ACOLUMN2					Default	SYSFMT8	Loca

4. Add three additional columns:

#### Sequence Type Caption Decimals Date/Time Output Format

- 30 AColumn1 Department
- 40 NColumn1 Salary 2
- 50 DColumn1 Start Date DDMMMCCYY

Note that you may set the initial width of each column as a percentage. Set these to suitable values.

								1
	Sequence	Type	Caption	Width % (Total 25%)	Decimals	Edit Code	Date/Time Output Format	UŢ
	10	VISUALID1	Number	25		Default	SYSFMT8	Lα
	20	VISUALID2	Full Name			Default	SYSEMT8	Lo
	30	ACOLUMN1	Department			Default	SYSEMT8	Lo
	40	NCOLUMN1	Salary		2	Default	SYSEMT8	Log
	50	DCOLUMN1	Start Date			Default	DDMMMCCYY	Loc
4	~~~~~~	ARQUUMNZ			horn	Default	WSEMTR ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	Lg

- 5. Close the business object properties dialog.
- 6. *Save and Exit* the Framework

# **Step 2. Change the Filter**

You need to make some simple changes to your filter to populate the new instance list columns.

- 1. Open the reusable part iiiVFW04 in the Visual LANSA editor. Note that you will find it on your *Favorites / Last Opened* tab.
- 2. Change the Group_by XG_IDENT to include fields DEPTMENT, SALARY and STARTDTE. Your code should now look like the following changes are shown in red.:

Group_By Name(#XG_Ident) Fields(#EMPNO #SURNAME #GIVENAME #

3. In the uSelectData method routine, within the SELECT/ENDSELECT loop, set up field VF_ELDTS to contain STARTDTE, converted to an ISO date and then converted to a display string.

The field VF_ELDTS is an alphanumeric field of length 19, which exists in the repository and is used by Framework components.

Your code should use intrinsic functions. For example:

#vf_eldts := #startdte.asdate( DDMMYY ).asDisplayString( ISO )

Use this code if your date format requires it:

#vf_eldts := #startdte.asdate( MMDDYY ).asDisplayString( ISO )

4. Change the invoke AddtoList method in the List Manager to populate the additional columns. For example:

Invoke Method(#avListManager.AddtoList) Visualid1(#UF_VisID1) Visualid2

- 5. Compile your filter.
- 6. Start the Framework and test the result.

# Summary

# **Important Observations**

• The Instance list supports up to 10 alpha and 10 numeric columns and up to 5 data columns.

# **Tips & Techniques**

• Instance list columns are shown, once you give them a sequence number.

# What You Should Know

• How to add columns to the instance list.

# VFW050 – Basic Combo Box Processing

# Introduction

The combo box is a simple list component. There are a number of other list components available such as **List View**, **Grid** and **Tree View**.

This exercise introduces a combo box (or dropdown) by adding it directly onto a panel and then adding logic to populate it with values and position it to an entry, when necessary. As you will see in a later exercise, you will often decide to create a reusable part which supports one or more linked combo boxes. This will simplify your application coding and enable the combo box logic to be written once and re-used.

A combo box typically displays a list of descriptions (such as Department Description). However your program logic will require the associated department code. Typically the combo box has one visible column containing descriptions and a hidden column containing the associated code.

When an entry in a list component is selected, your program variables are automatically populated from the selected row.

# Objective

• To enhance the Employee Details command handler (iiiVFW06) using a combo box.

								X
<i>屬</i> Employee : Details (A1)	003-SMITHE	Robert)						
Details 😥 Brief Notes	🚺 Images	Notes	😬 Skills 2 🛛 📍	Skills				
Employee Number	A1003			Termination Da	ate (DDMMYY)	0/00/00		Save
Employee Surname	SMITHE							Back
Employee Given Name(s)	Robert							
Street No and Name	29 Arthur Ro	ad,						
Suburb or Town	DEE WHY.							
State and Country	NSW.							
Post / Zip Code	2000							
Home Phone Number	977 6268							
Business Phone Number	406 6395			epartments				
FLEET ADMINISTRATION				INDO DOX				
Department Code	FLT		_					
Section Code	02							
Employee Salary	31,000.04							
Start Date (DDMMYY)	21/12/85							
					Durit I	I and I a		20242 4425
				Messages	Ready	Local	ING JIVORY12	7/08/12 14:05

To achieve this objective you will complete the following:

Step 1. Add a Combo Box to the PanelStep 2. Set up the Combo BoxStep 3. Test the Combo BoxSummary

# **Before You Begin**

Complete exercises VFW030, VFW040 and VFW042.

### Step 1. Add a Combo Box to the Panel

- 1. Open the reusable part iiiVFW06 in the editor.
- 2. In the *Design* view, select the *Controls* tab and drag and drop a *Combo Box* control onto the left hand panel (BODY_HEAD). Since BODY_HEAD has a flow down manager, your combo box will be positioned at the end of the existing fields.
- 3. Select the Department Code and select the *Details* tab. Note that this field has a *DisplayPosition* property of 10.
- 4. Select your combo box and:
  - a. change its *DisplayPosition* to 10 to position it immediately before the Department Code.
  - b. Change its *TabPosition* to 10.
- 5. Your panel should now look like the following:

	IIIVFW06 - Employee Details Comm
Employee Number	ABCDE
Employee Surname	aAbBcCdDeEfFgGhHiIjJ
Employee Given Name(s)	aAbBcCdDeEfFgGhHiIjJ
Street No and Name	aAbBcCdDeEfFgGhHiljJkKlLm
Suburbior Tavini	aAbBcCdDeEfFgGhHiIjJkklLm
State and Country	aAbBcCdDeEfFgGhHiljJkklLm
Rost / Zip Code	123456
Home Phone Number	ABCDEFGHIJKLMNO
Business Phone Number	ABCDEFGHIJKLMNO
Bepartment Code	ABCD

In a real application you would want the combo box positioned in line with the field edit boxes. This can be ignored for now. In later exercises you will see how this combo box would be implemented in a real application.

# **Step 2. Set up the Combo Box**

1. On the *Repository* tab, find the file DEPTAB and drag and drop the fields DEPTDESC and then DEPTMENT onto the combo box.

Adjust the Width of the Combo box so that the full description field is visible

2. Columns within a list component are components which have their own properties, events and methods. Review the code you have generated for the combo box:

Define_Com Class(#PRIM_CMBX) Name(#CMBX_1) Componentversion(1) Define_Com Class(#PRIM_CBCL) Name(#CBCL_1) Displayposition(1) Pare Define_Com Class(#PRIM_CBCL) Name(#CBCL_2) Displayposition(2) Pare

Component CBCL_1 is the first column of combo box CMBX_1. Note that CBCL_1 has a *Parent* of CMBX_1.

The combo box can have only one visible column: the first column added. If you added DEPTMENT first, correct the set up by making the column containing DEPTMENT *Visible* **false**.

In the *Design* view, to work with the combo box column 1, select the component CBCL_1 from the dropdown at the top of the *Details* tab.

Details		6	Design	Source Multilingual	Details Re	pository Help Cross
CBCL_1 - DEPTMENT, Department	Code, Alphanumeric(4)	-	IIIVFWO	6A - Employee De	tails CH	
Properties Events Methods			Employ	ee Number	ABCDE	
ComponentClassName	PRIM_CBCL		Employ	ee Surname	ABCDEF	GHIJKLMNOPQRST
ComponentPatternName	PRIM_CBCL		Employ	ee Given Name(s)	ABCDEF	GHIJKLMNOPQRST
ComponentTag			Street	No and Name	aAbBcCo	dDeEfFgGhHiIjJkKlLm
ComponentTypeName	PRIM_CBCL		Suburb	or Town	aAbBcCo	DeEfEaGhHiliikKi m
DisplayPosition	0		Chalter	ad Caraba	- Al-D-C	
Name .	CBCL_1		State a	nd Country	aAbbccc	JUEETE gGNHUJJKKILM
Owner	#IIIVFW06A		Post / Z	tip Code	123456	
Parent	#CMBX_1		Home P	hone Number	ABCDEF	GHIJKLMNO
SortDirection	Ascending		Rusines	e Phone Number	ABCDEE	CHI 1KI MINO
SortPosition	0		busiles		ADCOD	di tananino
SortType	Word		ABCDE	FGHIJKLMNOPQRST		
Source	#DEPTMENT		Depart	ment Code	ABCD	
🖉 Visible	False		Section	Code	AB	
Width	20		Employe	an Calacy	122 455	790.12
WidthType	Scaleable		Employ	ee balary	123,430	,703.12

3. Create an uInitialize method routine and add code to populate the combo box. The uInitialize method is already defined in the ancestor (VF_AC010).Your logic should clear the list CMBX_1 and select all valid department

codes and descriptions from the table DEPTAB and add them to your combo box.

Note that the CMBX_1 can be used as the Fields() parameter in I/O commands. Your code should look like the following:

Mthroutine name(uInitialize) Options(*REDEFINE) Invoke Method(#Com_Ancestor.uInitialize) Clr_List Named(#CMBX_1) Select Fields(#CMBX_1) From_File(deptab) Add_Entry To_List(#CMBX_1) Endselect Endroutine

4. Since the Employee Details command handler is reading and updating an employee you need to ensure the combo box displays the required department description.

Entries in a list components can be processed using the SELECTLIST/ENDSELECT loop.

5. Add the following logic in the uExecute method routine, after the FETCH employee record.

```
Define Field(#deptw) Reffld(#deptment)
#deptw := #DEPTMENT
Selectlist Named(#CMBX_1)
Leave If(#DEPTMENT = #deptw)
Endselect
#CMBX 1.currentItem.focus := true
```

6. How do you discover that the combo box has a *currentitem* which has a *focus* property?

In the *Design* view, select the combo box and use the context menu on the combo box component and select *Combo Box: CMBX_1 / Features*.

Home Phone Number	ABCDEFGHIJKLMNO	]
Business Phone Number	ABCDEFGHIJKLMNO	]
aAbBcCdDeEfFgGhHiIj]	Combo Box: CMBX_1	Features
Section Code	Delete Component	Help
how were	Conv Componer	- here and

The *Features help* panel will be displayed.

Features 🖸
CMBX_1
Definition     Class     PRIM_CMBX     List with an edit box     Extends
<ul> <li></li></ul>
Usage in IIIVFW06A

- 7. Expand the Properties to find *CurrentItem*. Note that the + symbol indicates that *CurrentItem* can be expanded.
- 8. Double click on PRIM_CBIT to show the properties, events and methods for the list column component.

Features	8
🗢 🗣 🔹 🖄 🖉	l 🕨 🔄 🖗
Primitive	
PRIM_CBIT	List item is an entry in a list, tree view c
Definition	
Class	
PRIM_CBIT	List item is an entry in a list, tree view o
🖃 🎼 Extends	
PRIM_OBJT	Base object, use it as the ancestor of c
🗄 🚽 Events	
🗄 📝 Methods	
🕀 📄 Properties	

9. Expand *Properties* to see the *Focus* property. Double click on the focus property to see the help for the CurrentItem.Focus Property.



10. Compile your component.

### Step 3. Test the Combo Box

1. Execute your *Framework as Designer*, search for employees by name, and select one in the instance list to display the Employee Details command handler.

The combo box should display the correct description for the employee's Department Code.

- 2. Select another employee to check the combo box value is changing.
- 3. Change the department code using the combo box.

Note that the Department Code value is changed.

You will be able to save the employee record, provided that the Section Code is valid for the new department.

### Summary

### **Important Observations**

- All list components such as combo box, grid, list view and tree view are handled in a similar way.
- The order of the data is controlled by the sort position and not the order the data is added to the list.
- If you change the *SortPosition* property for CBCL_1 equal to **1**, the departments will be displayed in ascending order.

# **Tips & Techniques**

- This exercise shows typical simple combo box processing. You must ensure that the combo box is filled and correctly positioned.
- If the department's combo box was being used for the New Employee command handler, you would need to initially populate the combo box and then position it to the first entry. For example by adding this logic:

```
Get_Entry Number(1) From_List(#CMBX_1)
#CMBX_1.CurrentItem.Focus := true
```

# What You Should Know

• How to implement a simple combo box.

# VFW052 – Build a Working List of Selected Items

# Objectives

- To implement a List View component showing a list of all employees
- To learn how to handle selected items in a list
- To show how to use the TRANSFORM_LIST Built-in function to write data from a working list to a temporary file.
- To develop a reusable part as the Reports business object's Weekly command handler. The application will have no real purpose except to include this exercise into the Framework. Initially the command handler will update a list column when an entry is selected or unselected. Total salary will be calculated for selected employees
- To enhance the application to maintain a working list of selected entries.
- To learn how you can maintain a working list of selected items, dynamically, meaning the working list is updated each time a selection in the list changes.
- To implement a static working list, which is populated with selected entries from the list only when a button is clicked.

Weekly					
Employee Number	Employee Surname	Employee Given Name(s)	Employee Salary	Selected	
0070	BROWN	VERONICA	50,125.00		[
0090	BLOGGS	FRED JOHN ALAN	20,045.91		
0193	SIMPSON	FRED	35,000.04		
0907	JONES	ANNE	34,213.04	YES	
1001	JONES	BEN	2,345.82	YES	
1002	SMYTHE	JOHN	25,000.04	YES	
1003	SMITHE	Robert	31,000.04		
1004	SMITHSON	PAUL	21,000.04		
1005	SMITHS	PETER	46,700.04		
1006	SMITHERS	JACK	25,000.04	YES	
1007	SNELL	GEORGE	26,780.04		
1008	SNEDDON	ALLAN	450,000.04		
1009	SNASHALL	DAMIAN	31,000.04	YES	
1010	PERRY	WILLIAM	60,000.04		
1011	PERRIN	CHRISTOPHER	25,000.04	YES	
1012	PAUL	PATRICK	26,456.04		
1013	PATTISON	GEORGE	78,977.04	YES	
1014	MOORE	NHOL	68,000.04		
1015	WOODS	BRADLEY	313,000.04		
1016	TURNER	JACK	22,000.04		
1017	NEAVE	GARY	25,600.04		
1018	ZACHARIA	PAUL	25,900.04		
1019	DICKENS	CHARLES	45,000.04		1
otal Salary	221,536.06	Dynamic Save Stati	c Save		

To achieve these objectives you will complete the following:

Step 1. Create the Weekly Command HandlerStep 2. Handle Selected ItemsStep 3. Build a Dynamic Working List of Selected ItemsStep 4. Build a Static Working List of Selected ItemsSummary

# Step 1. Create the Weekly Command Handler

1. Create a new *Reusable Part / Panel*:

Name: iiiVFW07

Description: Select Items from List

- 2. Change the Ancestor to VF_AC010.
- 3. Select the *Design* ribbon. Click on the *New Layout* button to add an attachment manager to the reusable part.



- 4. Drag and drop a *Panel* component at the **bottom** of the form.
- 5. Drag and drop a List View component into the **center** of the top area.
- 6. From the *Home* ribbon, open the *Layout Helper* tab from the Views dialog. Select your component iiiVFW07, for example click on the reusable part's title bar. On the *Layout Helper*, select the *Child Details* tab and ensure that PANL_1 is defined as **Bottom** and LTVW_1 is defined as **Center**.

💶 Layout Helper 🛛 💂 Reposit	ory   🚓 Outline   🚘 Details   🌟 Favorites   👉 💷
	면 <u>제</u>
Layout Managed Component	Ē
iiiVFW07	*
Layout	li X
ATLM_1	*
Children 🌒 🗙 🕘 🗙	Layout Manager Details Child Details As Child Details
CLTVW_1	
PAINL_I	Category Attachment -
	Top Left Center Bottom Right None
man man	

- 7. Save your component.
- 8. On the *Repository* tab, find the file PSLMST and drag and drop fields EMPNO, SURNAME, GIVENAME and SALARY into the List View.
- 9. On the *Repository* tab, expand *Fields* and drag and drop field STD_TEXTS into the *List View*.
- 10. Click on the column heading for STD_TEXTS and use the *Details* tab to change the *Caption* to **Selected**, and the *CaptionType* to **Caption**.
- 11. Select the list view and use the *Details* tab to ensure the *SelectionStyle* is **Multiple**.
- 12. Save your changes. Your design should look like the following.

	IIIVFW07 -	Select Items from Lis	st	×
Employee Number	Employee Surname	Employee Given Name(s)	Employee Salary	Selected
Jan ABCDE	aAbBcCdDeEfFgGhH	aAbBcCdDeEfFgGhHiljJ	123,456,789.12	aAbBcCd
4				•
4		·····		
4				•
4				•
4				

13. In the *Repository* create a field TOTSALARY (it may already exist). The field should be Packed, 15.2 with an edit code which shows a negative sign (for example, edit code L).

- a. Add the field TOTSALARY onto PANL_1 at the bottom of the main panel.
- b. Change TOTSALARY *MarginLeft* property to 100. Adjust the *Width* property as required.
- 14. Create an Initialize event handling routine for the List View. Add code to:
  - Clear the list view
  - Change STD_TEXTS to blanks
  - Select the required fields from file PSLMST
  - Add and entry to the list view
  - End Select.

Your code should look like the following:

Evtroutine Handling(#LTVW_1.Initialize) Options(*NOCLEARMESSAGES [•] Clr_List Named(#LTVW_1) #std_texts := *blanks Select Fields(#LTVW_1) From_File(pslmst) Add_Entry To_List(#LTVW_1) Endselect Endroutine

15. Save your changes.
# Step 2. Handle Selected Items

- 1. Create an ItemGotSelection event handling routine for the list view. Add code which:
  - Changes STD_TEXTS to YES
  - Updates entry in the list view
  - Add Salary to TOTSALARY
- 2. Create an ItemLostSelection event handling routine. Add code which
  - Changes STD_TEXTS to blank
  - Updates entry in the list view
  - Subtracts Salary from TOTSALARY.

Your code should look like the following:

Evtroutine Handling(#LTVW_1.ItemGotSelection) Options(*NOCLEARMES #std_texts := 'YES' Upd_Entry In_List(#LTVW_1) #TOTSALARY := #TOTSALARY + #salary Endroutine Evtroutine Handling(#LTVW_1.ItemLostSelection) Options(*NOCLEARMES #std_texts := *blanks Upd_Entry In_List(#LTVW_1) #TOTSALARY := #TOTSALARY - #salary Endroutine

- 3. Compile your component.
- 4. Execute the *Framework as Designer*.
  - a. Open the *Reports* business object properties dialog.
  - b. Select the *Commands Enabled* tab.
  - c. Select the *Weekly* action and define the Windows Component as iiiVFW07.
  - d. Close the *Properties* dialog
- 5. *Save and Restart* the Framework.
- 6. Test the *Weekly* command handler for the *Reports* business object.

- a. Click on entries using the shift or control key to select multiple entries. The Selected column should be updated to **YES** and the **Total Salary** field should reflect the selected employees.
- b. Click on white space below the employee entries to unselect all entries. **Total Salary** should now be blank (zero value).

# Step 3. Build a Dynamic Working List of Selected Items

In this step you will extend your Weekly command handler by defining a working list. The *ItemGotSelection* and *ItemLostSelection* event routine will be extended to add or delete entries to/from the dynamic working list.

A *Dynamic Save* push button will be added to PANL_1. When clicked this button will write a comma separated file from the dynamic working list.

- 1. Define a work field called KEYEMPNO which refers to field EMPNO for its definition. This will be used to store EMPNO in the working list
- 2. Define a work field RETCODE which refers to field IO\$STS for its definition.
- 3. Define a working list DYNAMIC containing fields KEYEMPNO, SURNAME, GIVENAME and SALARY. Specify the number of entries as *MAX. Define the list Counter as LISTCOUNT.

**Note:** *MAX denotes the maximum list sized is only limited by the execution platform. For *MAX, the working list uses memory dynamically.

Your code should look like the following:

Define Field(#keyempno) Reffld(#empno) Define Field(#retcode) Reffld(#io\$sts) Def_List Name(#dynamic) Fields(#keyempno #surname #givename #salary) C

- 4. Modify the ItemGotSelection event routine:
  - a. Look in the DYNAMIC working list (LOC_ENTRY) for the selected employee number. Use Where() to compare KEYEMPNO with EMPNO.
  - b. If not found, add an entry to list DYNAMIC. Be sure to first change field KEYEMPNO to the value of EMPNO (ADD_ENTRY).
- 5. Modify the ItemLostSelection event routine:
  - a. To look in the DYNAMIC working list for the selected employee number using LOC_ENTRY with a suitable Where() parameter.
  - b. If found, change KEYEMPNO to EMPNO and delete the entry from list DYNAMIC (DLT_ENTRY).
- 6. Add a push button to the bottom panel (PANL_1) and change the *Name* to **PHBN_DYN**. Change the *Caption* to **Dynamic Save**.

- 7. With the PANL_1 selected, select the *Design* ribbon..
  - a. Click on *New Layout* and click on *Flow Across* to add a flow across manager to this panel.

New Layout	Top	Bottom	Left				
New Layout							
Table Table Layout Manager							
🖸 💻 Flow	Flow Down						
😳 💳 Flow I	Flow Down Layout Manager						
Flow	<b>Across</b> Across Lay	out Mana	ger				
Split F	Split Horizontal Split Horizontal Layout Manager						
Split V	Split Vertical Split Vertical Layout Manager						

b. On the *Layout Helper* tab, click on the right hand blue paper clip button, to *Attach all children to the layout manager*.

💶 Layout Helper 🛛 💂 Reposito	ny   🛧 Outline   📜 Details   🚺
-	± ×
Layout Managed Component	Ē
PANL_1	
Layout	🗂 🗙
FWLM_1	· · · · · · · · · · · · · · · · · · ·
Children	Layout Manager Details Chile
V PHBN DYN	
V TOTSALARY	Category
	www.www.

- c. On the *Layout Manager Details* tab, select the *Category / Margins* and use the **All** setting to define all margins as 6 pixels.
- d. Your design should look like the following:



- 8. Create a Click event routine for the push button PHBN_DYN. Add logic to:
  - a. Check if LISTCOUNT is greater than zero
  - b. Use the TRANSFORM_LIST BIF to create a comma separated file. Place the in C:\temp. For example:

```
if (#listcount > 0)
```

Use Builtin(TRANSFORM_LIST) With_Args(#Dynamic 'C:\temp\iiiDynamic Endif

**Hint:** Use the *F4 Command Assistant* to complete the USE command.

c. Check if RETCODE is OK. Issue a "Dynamic file saved to . . . " message, or an error message if the save was not successful.

Your code should now look like the following. New / changed code is highlighted in red, italic.

```
Evtroutine Handling(#LTVW_1.ItemGotSelection) Options(*NOCLEARMES
#std texts := 'YES'
Upd Entry In List(#LTVW 1)
#TOTSALARY := #TOTSALARY + #salary
#keyempno := #empno
Loc_Entry In_List(#Dynamic) Where(#empno = #keyempno) Ret_Status(#
If (#retcode *NE OK)
Add Entry To List(#Dynamic)
Endif
Endroutine
Evtroutine Handling(#LTVW_1.ItemLostSelection) Options(*NOCLEARMES
#std texts := *blanks
Upd_Entry In_List(#LTVW_1)
#TOTSALARY := #TOTSALARY - #salary
Loc_Entry In_List(#Dynamic) Where(#empno = #keyempno) Ret_Status(#
If (#retcode *EO OK)
Dlt_Entry From_List(#Dynamic)
Endif
```

Endroutine Evtroutine Handling(#PHBN_DYN.Click) If (#listcount > 0) Use Builtin(TRANSFORM_LIST) With_Args(#Dynamic 'C:\temp\iiiDyna If (#retcode *EQ OK) Message Msgtxt('Dynamic file save to C:\temp\iiiDynamic.csv') Else Message Msgtxt('transform list failed') Endif Endif Endif

- 9. Compile your component
- 10. Execute the Framework and test your new Weekly command handler for the Reports business object. Check that the CSV file is saved with the correct contents. The file will open in Excel if available, otherwise open it using Notepad.

#### Notes:

If a path is not specified, the TRANSFORM_LIST BIF will save the output file in **<sysdir>.** 

**Optional:** You could have set the dynamic save push button to *Enabled(False)* initially and enable the button if LISTCOUNT is greater than zero. Add this logic to the *ItemGotSelection* and *ItemLostSelection* event routines.

## Step 4. Build a Static Working List of Selected Items

In this step you will extend the *Weekly* command handler, by defining a second working list, STATIC.

This list is called STATIC because when a button is clicked, the whole list view is read and currently selected items are selected, to build the STATIC list. Its entries are not maintained dynamically each time a list item gets or loses selection.

A *Static Save* push button click event will then process the List view adding only the selected items to the STATIC working list.

TRANSFORM_LIST will be used to save the STATIC working list as comma separated file.

- 1. Define a work field STATCOUNT with reference to field LISTCOUNT
- 2. Define a working list, named STATIC containing EMPNO, SURNAME, GIVENAME and SALARY. Set entries to *MAX and a Counter of STATCOUNT.

Your code should look like the following:

Define Field(#statcount) Reffld(#listcount) Def_List Name(#static) Fields(#empno #surname #givename #salary) Counter

- 3. Add a push button to the bottom panel (PANL_1). Change its name to PHBN_STAT and change *Caption* to **Static Save**.
- 4. Create a Click event routine for push button PHBN_STAT which:
  - a. Clears the static working list
  - b. Reads all entries in the List View using SELECTLIST/ENDSELECT.
  - c. Continues reading the next list view entry if the currentitem is not selected.

**Hint:** Use Feature Help (F2) on the list view and examine properties to find currentitem and then the properties of currentitem.

- d. For each selected item, add an entry to the static working list.
- e. If STATCOUNT is greater than zero, use the TRANSFORM_LIST BIF to save to a CSV file.
- f. Test RETCODE and issue a success or error message as appropriate.

```
Your code should look like the following:
Evtroutine Handling(#PHBN_STAT.Click)
Clr_List Named(#Static)
Selectlist Named(#LTVW_1)
Continue If(*Not #LTVW_1.currentitem.selected)
Add_Entry To_List(#Static)
Endselect
Use Builtin(transform_list) With_Args(#Static 'c:\temp\iiistatic.csv' O) To_Get
If (#retcode = OK)
Message Msgtxt('Static file saved to C:temp\iiistatic.csv')
Else
Message Msgtxt('Static file save failed')
Endif
Endroutine
```

## Summary

- List components have many other features than those in this exercise. Many of these will be explored in later exercises.
- A later exercise will look at how a working list can be passed to a function and received by a function.

## **Important Observations**

- LANSA list component processing is very fast. Even with thousands of entries, there will only be a short delay while creating the static saved file.
- Both static and dynamic methods work efficiently and are equally straight-forward to program.
- You should always review the properties, events and methods which are available for a control, to understand how you can use it to meet your application requirements.

# **Tips & Techniques**

- If the user occasionally needs to produce the save file, the static method would probably be the most suitable.
- If the user needs to see the total selected salary and frequently create a saved working list, the dynamic method would work well.

# What You Should Know

- How to handle list events.
- How to update entries in a list component.
- How to process all entries in a list component
- How to define and use a working list.
- How to use the TRANSFORM_LIST BIF.

# VFW054 – Edit Text in a Memo / Edit Box

## Objective

- To demonstrate how a multi-line edit box can be used
- To build the Employee *Brief Notes* command handler, which will use the Multi-line edit box component to create, save and display text for an employee. When an employee is selected in the instance list, the *Brief Notes* command handler will retrieve the text and populates the multi-line edit box
- To learn how to use *Save* and *Delete* buttons, enabling notes to be saved or deleted.
- To create a simple file to store the notes data.



To achieve these objectives you will complete the following:

Step 1. Create a Table to Store Employee Notes

Step 2. Create Brief Notes Command Handler

Step 3. Create the Command Handler Logic

Step 4. Implement Memo Box Copy/Paste Methods (Optional)

Summary

## **Step 1. Create a Table to Store Employee Notes**

Field Name	Description	Туре	Length	Decimals	Input Attrib.
iiiNTETYPE	Note Type	Alphanumeric	1		
iiiNTESQN	Note Sequence Number	Packed	7	0	
iiiLNECNT	Line Count	Packed	7	0	
iiiNTELNE	Note Line	Alphanumeric	80		LC

1. Define the following fields in the Repository:

2. Create a table iiiEmpNotes - Employee Notes, that is defined as follows:

Field	Primary Key Number
EMPNO	1
iiiNTETYPE	2
iiiNTESQN	3
iiiLNECNT	4
iiiNTELNE	

The file does not need to be RDMLX enabled.

3. Compile your new table. Be sure to select compile options which will build table, indexes and OAM.

## Step 2. Create Brief Notes Command Handler

1. Create a new Reusable Part / Panel:

#### Name: iiiVFW08

#### Description: Employee Brief Notes

- 2. Give the reusable part an ancestor of VF_AC010
- 3. Select the *Design* ribbon. Give the component an *Attachment manager*.
- 4. Drop a Panel onto the **right hand** side. Give this panel the *Name*, **BUTTON_PANL**. Adjust the width so that it can contain push buttons.
- 5. Drop a *Multi-line edit box* onto the **center** of the left hand area.
- 6. Open the *Layout Helper* tab. If necessary, select it from the *Views* button on *Home* ribbon.

Select iiiVFW08 in the *Layout Managed Component* dropdown and select the *Child Details* tab. Check the *Child Details* are correctly defined.

- a. BUTTON_PANL should be **Right**
- b. MEMO_1 should be **Centre**.
- c. Select the BUTTON_PANL and give it a *Flow Down* manager.
- d. On the *Layout Manager* tab. On the *As Child Details* tab, select the *Category / Margins* and use the *All* to set margins of **6** pixels.
- 7. Drop two push buttons onto the BUTTON_PANL.
  - a. *Name* the first **PHBN_SAVE** with a *Caption* of **Save**.
  - b. *Name* the second **PHBN_DLT** with a *Caption* of **Delete**.
- 8. Save your changes.
- 9. Locate your Employee Notes file on the *Repository* tab.

Drop the field iiiNTELNE onto the Memo box.

The Multi-line edit box has columns just like other list components. You have just created column MECL_1.

10. Drop the field iiiNTESQN onto the Memo box.

At this point the new column MECL_2 should be selected. Use the *Details* tab to check its *ColumnRole* property. Since this is a numeric field it should have been automatically set to *LineNumber*.

The Memo box will set this line number column as each line is created. This enables application code to be simplified, when writing the note data to a file. The sequence number will ensure the text is restored to the memo box in the same sequence.

- 11. Drop field iiiLneCnt onto the Memo box. Once again check this column's *ColumnRole* which should be **LineContinuation**.
- 12. Select the Memo box and set its properties as follows

Property	Value
WordWrap	True
AddEntryMode	MultiplePerLine
MaximumLineLength	20,000

The Memo box component will increment line count (iiiLneCnt) as text is added.

## **Step 3. Create the Command Handler Logic**

The *Brief Notes* command handler is an instance list command. These means it should begin by retrieving the instance list current entry, so that it can restore notes for this employee.

- 1. Create an uExecute method routine with an Options(*redefine) parameter. This routine needs to do the following:
  - a. Invoke the ancestor **u**Execute method
  - b. Invoke the List Manager GetCurrentItem method to retrieve AKey1 (contains EMPNO)
  - c. Clear the Memo box
  - d. Select all iiiEmpNotes records for this EMPNO and iiiNteType = G
  - e. Add entries to Memo box

Your code should look like the following:

```
Mthroutine Name(uExecute) Options(*redefine)
#com_ancestor.uExecute
#avlistmanager.getCurrentInstance Akey1(#empno)
Clr_List Named(#MEMO_1)
#iiiNTETYP := G
Select Fields(#MEMO_1) From_File(iiiEmpNotes) With_Key(#empno #iiiNte
Add_Entry To_List(#MEMO_1)
Endselect
Endroutine
```

2. In this step you will add logic to handle the *Save* button click event.

The Memo box has a *Modified* property, which your save logic can test to determine whether the notes data needs to be saved. Modify your uExecute routine so that the memo box *Modified* property is set to false, before clearing the Memo box. For example:

#MEMO_1.modified := false

Remember you can find out more about any component's properties, event and methods by using the *F2 Feature Help*.

Create a Click event handling routine for the *Save* push button. Add logic to perform the following:

If the Memo box Modified property is true – perform

Change iiiNTETYPE to G

Delete all records from iiiEmpNotes for this employee and Note Type.

Read all items in the Memo box using SELECTLIST

Insert new records to iiiEmpNotes

Your code should look like the following:

Evtroutine Handling(#PHBN_SAVE.Click) If (#MEMO_1.modified) #iiiNteType := G Delete From_File(iiiEmpNotes) With_Key(#empno #iiiNteType) Selectlist Named(#MEMO_1) Insert Fields(#iiiNTELNE #iiiNteType #iiiNTESQN #iiiLneCnt #empno) To_I Endselect Endif Endroutine

3. Create a Click event handling routine for the *Delete* button. Add logic to delete all entries for this employee with iiiNTETYPE = G.

Clear the memo box.

Your code should look like the following:

```
Evtroutine Handling(#PHBN_DLT.Click)
#iiiNteType := G
Delete From_File(iiiEmpNotes) With_Key(#empno #iiiNteType)
Clr_List Named(#MEMO_1)
Endroutine
```

- 4. Compile your component.
- 5. Execute the *Framework as Designer* and open the properties dialog for the

Employees business object.

- 6. Select the *Commands Enabled* tab, select the *Brief Notes* action and define its *Windows command handler* as **iiiVFW08**.
- 7. *Save and Restart* the Framework as an end user.
- 8. Test the *Brief Notes* command handler.
- 9. Using standard Windows shortcuts (Copy = Ctrl+C, Paste = Ctrl+V), copy text from a portion of the LANSA Online Guides and paste it into employee brief notes.

# Step 4. Implement Memo Box Copy/Paste Methods (Optional)

If you examine the Memo box *Methods* you will find it supports *Copy*, *Paste*, *Cut*, *Print* and *Find*.

*Copy* will copy selected text to the Windows clipboard. The method can be actioned programatically or by the user using the standard shortcut keys, such as Ctrl+C

Note that there are other Methods available such as *Replace*, which you should investigate later. Once again Feature Help (F2) will provide more information.

In this step you will implement some of these methods using a Pop-Up menu.

1. In the *Design* view, select the Memo box, select *All Controls* on the *Controls* tab and drag and drop a *Pop-up Menu* component onto it. The *Pop-up menu* will be displayed at the top of the *Design* tab.



You create can Menu items by typing their *Caption*, then pressing *Enter* to create the next menu item.

The first *Pop-up menu* component is named PMNU_1, and *Menu items* are named MITM_1, MITM_2 etc.

A component, such as the *Memo box*, will have its *Popupmenu* property set to the name of the *Pop-up menu* which was dropped onto it.

Typing a dash into a menu *Caption* makes this menu item a *divider*. Its purpose is to visually separate different parts of the menu.

- 2. In this step you will define the *Pop-up menu items*. If the first menu item (Item1) is not displayed at the top of the *Design* tab, use the context menu on the Memo box to *Edit pop-up menu*.
  - a. Replace the Item1 text with Copy and press *Enter*.
  - b. Enter Paste and press Enter.
  - c. Type a dash (-) character into the next menu item, and press enter
  - d. On the new menu item, type Select All and press Enter.
  - e. Type dash into the next menu item and press Enter.

- f. Type Find into the new menu item
- g. Save your changes.

Your *Pop-up menu* should now look like the following:

Design Source Mult	tilingual Details   Reposito
Сору	rief Notes
Paste	_mMnNoOpPqQrRsSt
Select All	
Find	
the man	

3. Create a Click event handling routine for each menu item. To do this, simply position in each menu item and use the *Details / Events* tab.

**Note** that you could have renamed each menu item (for example, MITM_COPY) for example. This is recommended in your own applications. It will make future maintenance much easier.

4. Complete each menu item *Click* event so that it invokes the relevant Memo box component method. For example:

Evtroutine Handling(#MITM_1.Click) #MEMO_1.copy Endroutine

- 5. Compile your component.
- 6. Execute the Framework as End User and test your *Brief Notes* command handler and the *Pop-Up Menu* functions.

For example you could copy and paste within the same Employee's notes or switch to another employee's *Brief Notes* to copy into that one.

Test the Copy, Paste, Select All and Find functionality.

#### Summary

#### **Important Observations**

- The Copy / Paste / Cut etc functionality of the memo box can also be invoked using the standard Windows short cut keys: Ctrl+C, Ctrl+V and Ctrl+X.
- This simple example uses a line length of 80 characters, which would be restrictive if you need to copy a large piece of text from an existing document.

## **Tips & Techniques**

• The multi-line edit box will handle large blocks of text and word wrap.

# What I Should Know

- How to implement text input in a multi-line edit box.
- How to implement memo box functions using a pop-up menu.

## VFW056 – Process a List in Sorted Order

### Objective

- To demonstrate how to process a list in either loaded or sorted order.
- A *Reports / Sort* command handler will have two List views. The left hand list view displays all employees, with sorting by column enabled. A second List view displays selected items in loaded or sorted order.

Sort	Weekly									
Employee	Employ	Employee	Departmen	Section Code		Employee	Employee	Employee	Departme	Section C.
1031	BLAKE	JOHN	MIS	EI		A1031	BLAKE	JOHN	MIS	EI
0090	BLOGGS	FRED JOH	ADM	09		A0090	BLOGGS	FRED JO	ADM	09
40070	BROWN	VERONICA	AUD	02		A0070	BROWN	VERONICA	AUD	02
A2001	BROWN	HARRY GE	ADM	05		A2001	BROWN	HARRY G	ADM	05
13564	BROWN	FREDDY	ADM	04		A3564	BROWN	FREDDY	ADM	04
42002	BROWNLOW	ARTHUR	ADM	04		A2002	BROWNL	ARTHUR	ADM	04
A1019	DICKENS	CHARLES	LEG	01		A1019	DICKENS	CHARLES	LEG	01
A1020	DOUGLAS	ADAM PETER	ADM	01	=	A1020	DOUGLAS	ADAM PE	ADM	01
1234	JACKSON	STEPHEN	SD	ES		A1234	JACKSON	STEPHEN	SD	ES
0907	JONES	ANNE	AUD	03		A0907	JONES	ANNE	AUD	03
1001	JONES	BEN	GAC	01		A1001	JONES	BEN	GAC	01
42000	JONES	JAMES	ADM	02						
A2003	JONES	JAMES	ADM	04						
1026	LEWIS	TONY	GAC	03						
1032	LINCOLN	PAUL	INF	03						
41028	MAXWELL	ANDREW	GAC	FC						
1021	MCCULLY	DAVID	SD	ES						
1014	MOORE	JOHN	ADM	02						
1027	MORRISON	ALAN	SD	ES						
1404	MRS BRICK	GILL	ADM	01						
1017	NEAVE	GARY	INF	02						
41013	PATTISON	GEORGE	SD	01						
41012	PAUL	PATRICK	GAC	01						
1011	PERRIN	CHRISTOP	AUD	01						
1010	PERRY	WILLIAM	AUD	03						
A1509	REDFORD	ROBERT	ADM	01	-					

In order to achieve these objectives you will complete the following:

Step 1. Create Sorted Command Handler

Step 2. Complete the Command Handler logic

Summary

## **Before You Begin**

Complete exercises VFW030, VFW040 and VFW042

## Step 1. Create Sorted Command Handler

1. Create a new reusable part:

#### Name: iiiVFW09

#### Description: Loaded or Sorted List Items

- 2. Change the component's *Ancestor* to VF_AC010.
- 3. Select the *Design* ribbon and give the reusable part an Attachment manager.
- 4. Drop a panel onto the bottom and changes its name to BUTTON_PANL
- 5. Drop a Panel onto the centre and change its name to MAIN_PANL.

If necessary, select IIIVFW09 in the *Layout Helper* tab and ensure the Child Details are correctly defined.

- 6. Select the MAIN_PANL. On the *Design* ribbon, select *Split Vertical* on the *New Layout* menu to give the panel a *Vertical Split layout manager*.
- 7. Use the *Layout Helper /Layout Manager Details / Category / Divider Style* to change the *Divider Style to* **Raised**.

The *Vertical Splitter* creates two new panels. Rename these as MAIN_LEFT and MAIN_RIGHT as follows:



8. Select panel MAIN_LEFT, and on the *Layout Helper* tab give it an attachment manager of ATLM_1. Do this by selecting the attachment manager ATLM_1 in the *Layout dropdown*.

Layout Helper	
Layout Managed Component	Ē
MAIN_LEFT	-
Layout	🛎 🗙
ATLM_1	*
Children 🔋 🗙 🔋 🗙	Layout Manager Details Child D
	Category Processing Order

- 9. Give the MAIN_RIGHT panel the ATLM_1 attachment manager.
- 10. Drop a list view into the **center** of MAIN_LEFT and rename the list view **LIST_1**.
- 11. Drop a list view into the **center** of the MAIN_RIGHT panel and rename the list view **LIST_2**.
- 12. Save your changes.
- 13. Select the file PSLMST on the *Repository* tab and drag fields EMPNO, SURNAME, GIVENAME, DEPTMENT and SECTION into each list view. In a real application you would spend some time making the column headings suitable.
- 14. Select list view LIST_1 and select each column. Do this by clicking on each column heading and changing the column's *SortOnClick* property to **True**.
- 15. Create an *Initialize* event routine for LIST_1 and add code to populate it with all records from the file PSLMST. Your code should look like the following:

Evtroutine Handling(#LIST_1.Initialize) Options(*NOCLEARMESSAGES * Clr_List Named(#LIST_1) Select Fields(#LIST_1) From_File(pslmst) Add_Entry To_List(#LIST_1) Endselect Endroutine

- 16. In the *Layout Helper*, select the BUTTON_PANL and give it a *Flow Across manager*. On the *Layout Manager Details* tab, select *Margins* and use the **All** setting to set all *Margins* to **6** pixels.
- 17. Add a push button to the BUTTON_PANL and change its name to PHBN_LOADED.

- a. Change its *Caption* to **Loaded Order**.
- b. Create a *Click* event routine.
- 18. Add a second push button to the BUTTON_PANL.
  - a. Change its *Name* to **PHBN_SORTED**.
  - b. Change its *Caption* to **Sorted Order**
  - c. Create a *Click* event routine.
- 19. Save your changes.

## Step 2. Complete the Command Handler logic

- 1. Add code to the PHBN_LOADED click event routine, which:
  - a. Clears the list view LIST_2
  - b. Selects all items in LIST_1 using SELECTLIST
  - c. If LIST_1 current item, selected is true, add an entry to LIST_2

For example:

Evtroutine Handling(#PHBN_LOADED.Click) Clr_List Named(#LIST_2) Selectlist Named(#LIST_1) If (#LIST_1.currentItem.selected) Add_Entry To_List(#LIST_2) Endif Endselect Endroutine

- 2. Add code to the PHBN_SORTED *Click* event routine which:
  - a. Clears the list LIST_2
  - b. Processes all items in LIST_1 using a FOR/ENDFOR loop. For example:

```
Evtroutine Handling(#PHBN_SORTED.Click)
Clr List Named(#LIST 2)
```

```
For Each(#row) In(#LIST_1.items)
```

```
....
Endfor
Endroutine
```

LIST_1.Items returns a reference to the *Items* collection in the list view component.

Within the FOR/ENDFOR loop, each item will be referred to as #ROW

c. Retrieve each list item using GET_ENTRY. For example

Evtroutine Handling(#PHBN_SORTED.Click) Clr_List Named(#LIST_2) For Each(#row) In(#LIST_1.items) Get_Entry Number(#row.entry) From_List(#LIST_1) Endfor Endroutine

Row.Entry is the entry number for each list item

d. If the item is selected, add an entry to LIST_2. For example

Evtroutine Handling(#PHBN_SORTED.Click) Clr_List Named(#LIST_2) For Each(#row) In(#LIST_1.items) Get_Entry Number(#row.entry) From_List(#LIST_1) If (#row.Selected) Add_Entry To_List(#LIST_2) Endif Endfor Endroutine

- 3. Compile your command handler.
- 4. Execute the *Framework as a Designer* and open the properties dialog for the *Reports* business object and select the *Commands Enabled* tab.

Select the *Sort* action and define its *Windows command handler* component as iiiVFW09.

- 5. *Save and Restart* the Framework.
- 6. Test the *Sort* command handler for the *Reports* business object.
  - a. Right click on the *Weekly* command tab and select the *Sort* action from the context menu.

Note that initially the left hand employee list is displayed in its loaded order.

- b. Without sorting on a column in the left hand list, select some entries and use the *Loaded Order* button. The right hand list should now contain the selected entries as they are displayed on the left, which is their loaded order.
- c. Test the results using the *Loaded Order* and *Sorted Order* buttons. The right hand list should contain the same selected entries in each case.
- d. Sort the left hand list on Surname. Select entries and test the *Loaded Order* button. Note that the right hand list contains entries from their

loaded position, which is no longer the same as the displayed list.

e. Now try the *Sorted Order* button and note that the right hand list now contains the same entries as displayed and selected in the left hand list. The list has been processed in its sorted order.

### Summary

#### What You Should Know

- How to enable a list's columns to allow sorting.
- How to process a list's items in loaded order.
- How to process a list's items in sorted order.
- How to use the FOR/ENDFOR loop.

## VFW060 – Using a Tree View

### **Introduction to Tree Views**

Tree view components are widely used in Windows applications. They usually present related information, with the tree visually representing the relationships as levels. For example folder, sub folders and files; or departments, sections and employees.

The VL Framework usually presents the navigation panel as a tree (it can in fact also be visualized as two lists or as a dropdown.

# The Tree View Technically

The tree view component is specialized list component, which presents the list items as a tree.

The Visual LANSA Tree View component, by default has a *ViewStyle* property of **Levelled**.

In this case, each level can display one column only.

Each column added must have its correct Level number defined.

Each level must have a column which defines the KeyPosition for this level. For example, if level one is department description, then level one probably has DEPTDESC defined as *KeyPosition* = **1**. It defines the sequence of entries for level one. For example:



The above tree would be defined as follows:

#### SourceField Level KeyPosition

DEPTDESC11SECDESC21FULLNAME 31

- Note that the order added is not related to how an entry is visualized in the tree view.
- Each value of DEPTDESC is displayed once, at level 1 and in sequence.
- If the tree is to show all departments from the table DEPTAB and all sections for each department from the table SECTAB and all employees for each section, then it is essential to load:
  - a. the tree view entries from DEPTAB
  - b. all section's entries for each department from SECTAB
  - c. all employees for department and section from view PSLMST1.

# Objective

- To build a simple three level tree view for all departments, sections and employees
- To build tree view with columns
- These exercises will be implemented as a standalone form, not within your Framework application.

In order to meet the objectives you will complete the following:

Step 1. Create Tree View Form
Step 2. Displaying Tree View Data.
Step 3. Add Fields to Tab Sheets and Item Got Selection logic.
Step 4. Fill the Tree View on Demand
Step 5. Add Icons to the Tree View
Summary

### **Before You Begin**

Complete earlier list exercises, VFW050 and VFW052.

### **Step 1. Create Tree View Form**

1. Create a new *Basic Form*:

Name: iiiVFW10

#### Description: Tree View and Details

- 2. On the *Controls* tab, select *All Controls* and add a *Status bar* to the form. This component is always attached to the bottom of a form. LANSA messages from the OAM or program Message commands are routed to the status bar.
  - ABCDEFGHJIKLIMMnNoOpPqQrRsStTuUvvvvXxYvzZ:
- 3. Add a *Tree view* to the left hand side of the form and resize it as shown:

4. On the *Repository* tab, select the DEPTAB file and add field DEPTDESC to the tree view.

Define its *KeyPosition* property as **1**.

- 5. Select the SECTAB file and add field SECDESC to the tree view. Change its *KeyPosition* to **1**.
- 6. Find field FULLNAME in the *Fields* section of *Repository* and add it to the tree view. Change its *Keyposition* to **1**.
- 7. Save your changes.
- 8. Extend the form's Initialize event routine to populate the tree based on the following pseudo code:

Clear the Tree view

Select all entries from file DEPTAB

Select entries for each department from file SECTAB

- Select entries from logical file PSLMST1 for each department/section
- Set up Fullname from fields Surname and Given Name

Your code should look like the following:

Evtroutine Handling(#com_owner.Initialize) Set Com(#com_owner) Caption(*component_desc) Clr_List Named(#TRVW_1) Select Fields(#deptment #deptdesc) From_File(deptab) Add_Entry To_List(#TRVW_1) Select Fields(#section #secdesc) From_File(sectab) With_Key(#deptment) Add_Entry To_List(#TRVW_1) Select Fields(#surname #givename) From_File(pslmst1) With_Key(#deptment #fullname := #surname + ', ' + #givename Add_Entry To_List(#TRVW_1) Endselect Endselect Endselect Endselect Endroutine

- 9. Compile and test your form. Your tree view should be populated with all departments, sections and employees.
- 10. In a later step you will consider how to fill the tree view 'on demand'.

## Step 2. Displaying Tree View Data.

In this step you will make design changes to the form, so that it will be able to display data for each level when a tree item is selected.

To achieve this, a number of issues must be addressed:

- Each level in the tree must contain key fields which will enable the additional data to be read from a file. You will add hidden columns to the tree view for each level.
- The form must be able to display either department, section or employee details. A *Tab Folder* and *Tab sheet* components will be added to the form to display the data for each level.
- Your 'display details' logic must know which level in the tree was selected
- Additional fields must be populated when the tree is built.
- 1. Open your form in the *Design* view and add the field DEPTMENT to the tree. Initially the tree will show this as a fourth level. Change its *Level* property to **1**. Change its *Visible* property to **false**.
- 2. Add field SECTION to the tree view. Changes its *Level* property to **2** and *Visible* to **false**.
- 3. Add EMPNO to the tree view. Change its *Level* to **3** and *Visible* to **false**.
- 4. Save your changes.
- 5. Expand the width of the form and add a *Tab Folder* component to the right hand side and resize it.

**Hint:** If you want to move the tab folder, click on the area to the right of the tabs, to select the tab folder component and then drag it.

6. Right click on the *Tab Folder* and use the context menu to *Add Page*. This will add a third tab sheet component inside the *Tab Folder*. Change its *Name* to **Sheet_3**.

You now need to be aware that there are components at two levels. The *Tab Folder* is a container. The *Tab Folder* is the *parent* of the *Tab Sheets*.

- a. Right click on the background area (just to the right of the "Page 1" tab and use the context menu to *Add Page*.
- b. Add a third page (or tab sheet).

Your design should look like the following:



- 7. Select the Page 1 tab and then click in the **center** to select the tab sheet itself (Sheet_1). Change its *Caption* to **Department Details**.
- 8. Repeat step 7, to change Sheet_2's *Caption* to **Section Details** and Sheet_3's *Caption* to **Employee Details**.
- 9. Save your changes.

## Step 3. Add Fields to Tab Sheets and Item Got Selection logic.

1. Find the DEPTAB file on the *Repository* tab and add fields DEPTMENT and DEPTDESC to the **Department Details** tab. Review the Notes below before you continue:

#### Notes:

If the *Layout Manager* is not being used (as in this exercise), the fields will be manually positioned. A good approach is to select both fields (hold down the Shift key to do so) and drag and drop them together onto the tab sheet. They will then be positioned, in line and one under the other.

The *Edit / Align* dialog enables you to position fields more accurately, when you are not using a *Layout Manager*. If a number of fields (or other components) are selected, the *Align* dialog will position them relative to the first field selected.

2. Find the SECTAB file and drag and drop fields DEPTMENT, SECTION, SECDESC, SECADDR1, SECADDR2, SECADDR3, SECPCODE and SECPHBUS onto the **Section Details** tab sheet.

**Note:** The DEPTMENT field will be renamed to DEPTMENT_1 as DEPTMENT already exists on this form.

**Hint:** If your tab sheet looks like this:



Select a field and change its *Width* property on the *Details* tab. For example *Width* = **350**.

3. Find the file PSLMST and drag and drop fields EMPNO, SURNAME, GIVENAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, SALARY, STARTDTE and TERMDATE onto the Employee Details tab sheet.

**Hint:** To drag the fields as one group: Select the first group of fields using the Shift key and then use the Control key to select the remainder (SALARY and so on).

- 4. Adjust field widths as necessary.
- 5. Save your changes.
- 6. Change the form *Initialize* event routine. Add field EMPNO to the fields retrieved from file PSLMST1.

Your code should now look like the following. The changed line is highlighted in red.

Evtroutine Handling(#com_owner.Initialize) Set Com(#com_owner) Caption(*component_desc) Clr_List Named(#TRVW_1) Select Fields(#DEPTMENT #DEPTDESC) From_File(deptab) Add_Entry To_List(#TRVW_1) Select Fields(#SECTION #SECDESC) From_File(sectab) With_Key(#DEPTN Add_Entry To_List(#TRVW_1) **Select Fields(#EMPNO #SURNAME #GIVENAME) From_File(pslmst1) '** #fullname := #SURNAME + ', ' + #GIVENAME Add_Entry To_List(#TRVW_1) Endselect Endselect Endselect Endselect Endselect

7. Create an *ItemGotSelection* event handling routine for the tree view.

The logic performed will need to be based on which level in the tree was selected.

The tree view *CurrentItem* component has a *Selected* property which enable the correct logic to be executed for each level.

**Important:** Remember that *Feature Help* (F2) will enable you to discover in detail the *Properties, Events* and *Methods* for any component.

The details for each level selected will be displayed by making the required tab sheet the "open" tab sheet. The Tab Sheet component has an *Opened* property. When **true** this tab sheet will be the "top" tab sheet and therefore

the required details will be visible.

Your code should look like the following:

```
Evtroutine Handling(#TRVW_1.ItemGotSelection) Options(*NOCLEARMES
#std_num := #TRVW_1.CurrentItem.level
Case (#std_num)
When (= 1)
#Sheet_1.opened := true
When (= 2)
Fetch Fields(*all) From_File(sectab) With_Key(#DEPTMENT #SECTION)
#DEPTMENT_1 := #DEPTMENT
#Sheet_2.opened := true
When (= 3)
Fetch Fields(*all) From_File(pslmst) With_Key(#EMPNO)
#Sheet_3.opened := true
Endcase
Endroutine
```

**Note:** The field DEPTMENT_1 is set to DEPTMENT for the Section details panel.

8. Compile and test your form.

You should be able to select any entry, and the relevant tab sheet will be displayed.

**Note:** You will probably notice that this form could be refined. For example if you click on the *Employee Details* tab while displaying *Section Details*, the *Employee Details* tab will contain values for the last employee. The *Opening* event for Sheet_2 could be used to clear fields on Sheet_1 and Sheet_3.
## Step 4. Fill the Tree View on Demand

The tree view is currently loaded with all department, section and employee data initially. Clearly this is an approach which will work well for a small amount of data. For your own applications, it is much more likely that you will need to populate the tree view "on demand". Your logic would add all records to level one, and then add to level two and three when a level is expanded (by click on the + next to that item).

In this step you will change your logic so that the tree view is populated on demand.

1. In the *Design* view, select the tree view and change its *ManageChildren* property to **true**.

The help for this property (see *F2 Features help*) will inform you:

ManageChildren controls what happens when a tree item is collapsed.

The *ManageChildren* property controls whether child items are automatically deleted when a tree item is collapsed. It will typically be used when the children are loaded during an *ItemExpanding* event. It can be set to True or False.

- 2. Create an *ItemExpanding* event handling routine for the tree view.
- 3. Cut and paste the Select sections and select employees logic from the from the *Initialize* event into the Tree view *ItemExpanding* event handling routine.

The form *Initialize* logic should now only add department data to the tree initially.

Your form *Initialize* should look like the following:

```
Evtroutine Handling(#com_owner.Initialize)
Set Com(#com_owner) Caption(*component_desc)
Clr_List Named(#TRVW_1)
Select Fields(#DEPTMENT #DEPTDESC) From_File(deptab)
Add_Entry To_List(#TRVW_1)
Endselect
Endroutine
```

4. Like the *ItemGotSelection* event routine, the *ItemExpanding* event will need to perform the action required based on the **level** selected. Add the required logic re-using the code pasted from the form *Initialize* routine.

- Assign TRVW_1.currentItem.Level to STD_NUM
- Within a Case loop for STD_NUM:
  - For Level 1, select from the sections file for this department
  - For Level 2, select from the employees file for this department / section.
  - For Level 3, no action required.

Your code should now look like the following:

```
Evtroutine Handling(#TRVW_1.ItemExpanding) Options(*NOCLEARMESS/
#std num := #TRVW 1.currentItem.level
Case (#std_num)
When (= 1)
Select Fields(#SECTION #SECDESC) From_File(sectab) With_Key(#DEPTN
Add_Entry To_List(#TRVW_1)
Endselect
When (= 2)
Select Fields(#EMPNO #SURNAME #GIVENAME) From_File(pslmst1) Wit
#fullname := #SURNAME + ', ' + #GIVENAME
Add_Entry To_List(#TRVW_1)
Endselect
When (= 3)
* no action required
Endcase
Endroutine
```

5. Compile your form and test it. From an end user perspective it should work exactly the same as before.

## Step 5. Add Icons to the Tree View

Tree views are usually shown with an icon displayed at each level. The image does not necessarily represent the level's data, but it gives a clear visual clue for the level being selected for example.

In fact the Tree View column has an *Image* and *ImageExpanded* property, so that two images can be associated with that level. Of course the image properties must be defined for the column which is displayed for that level.

Check your *Repository* under *Resources / Icons*. The following *Icon* components should already be defined:

Name	Use
VI_DEPTCL	Department (normal)
VI_DEPTOP	Department (open/expanded)
VI_SETCL	Section (normal)
VI_SECTOP	Section (open/expanded)
VI_EMPLOY	Employee

1. The *Image* properties need to be assigned at run time.

Add the following code to the beginning of the form *Initialize* event routine:

Set Com(#TVCL_1) Image(#vi_deptcl) Imageexpanded(#vi_deptop) Set Com(#TVCL_2) Image(#vi_sectcl) Imageexpanded(#vi_sectop) Set Com(#TVCL_3) Image(#vi_employ)

These assignments must be before departments are added to the tree view.

2. Compile and test your form, which should now look like the following:

Administration	<ul> <li>Department Details Section</li> </ul>	on Details Employee Details
	Employee Number Employee Surname Employee Given Name Street No and Name Suburb or Town State and Country Post / Zip Code Employee Salary Start Date (DDMMYY) Termination Date (DDM	A1001 Jones Shirley 144 Frog Lane Pymble NSW 2001 2,345.82 010288 MYY] 000000

3. Confirm that the icons change as a level is closed or expanded.

#### Summary

#### **Important Observations**

- All Visual LANSA list components control how the data is presented. The tree view displays a list as levels.
- The tree view definition controls how the data is visualized. The order in which the data is added is not important.
- A simple tree view (ViewStyle = unlevelled) displays one column for each level.
- One or more columns must be defined as the *Key Position* for each level.
- In this example the icons defined for each column (level) are fixed. If required, your application could set image properties after an entry is added. For example, use F2 Feature Help on a tree view component and drill down to examine *#TRVW_1.CurrentItem.Image*.

## **Tips & Techniques**

- A tree view which could contain a large number of entries should be filled on demand.
- The *ManageChildren* = **True** property, deletes entries for the next level, when a level is closed.
- All Visual LANSA list components are capable of holding many thousands of entries. However, from an efficiency and usability perspective, your application should control how much data is added to the list component.
- Icon components are compiled into the application.
- If the image file associated with an icon component is changed, you must recompile the application.

## What You Should Know

- How to implement a simple tree view (*ViewStyle* = **levelled**).
- How to manage displaying detailed data for a selected level.
- How to fill the tree view on demand.

#### VFW062 – A Tree View with Columns

- A Tree View with a property of *ViewStyle(Unlevelled)* may have multiple columns.
- *Unlevelled* means that the tree no longer creates a level per field, and instead, creates a column per field.
- The developer is now responsible for the level at which a tree item exists. This is governed by setting the *ParentItem* property of a tree item to another tree item.
- The tree view will have one set of columns and your logic must initialize and populate these as appropriate for each level in the tree.
- A new tree item's parent must be set to the relevant parent. A null parent means that the item will appear as a root node.
- The parent is no longer governed by the data. It is a choice the developer can now make.
- Effectively, there is NO LIMIT to the number of levels.

**Note:** The parent of a tree item is completely dynamic. It can be set at any time

## Objective

- To understand how a tree view can be defined and managed so that it displays columns as well as a tree view
- Your finished form will look like the following:

Code	Description	Address	Zip Code	Bus. Phone	Start Date	
ADM	Administration					
	Internal Admin	125 Main St., Blacktown	2167	679 2536	04 100 H 000	
A1001	Jones, Shirley	144 Frog Lane, Pymble,	2001	798 0543	01/02/1988	
A1012	Paul, Patrick	6 Camilo Avenue, Seve	214/	222 2222	01/05/1986	
A1013	Pattinson, George	12 Augusta Avenue,, Pu	2016	212 3569	01/12/1985	
A1015	Woods, Bradley	59 Darley Road, Bexley,	2030	789 4562	12/12/1984	
A1020	Douglas, Adam	6 Reading Avenue,, Kin	2147	639 5188	01/02/1988	
A1021	McCully, Lisa	15 Baker Place,, Pensh	2153	159 6845	01/03/1980	
A1025	Hobinson, Mary	14 Whitby Hoad,, St. Ive	2005	456 1852	01/05/1986	
A1027	Morrison, Alan	47 Lincoln Street,, Stan	2007	489 2485	01/02/198/	
A1111	Verey, Warren	1 Main Rd, Hill Lop, NSW	2345	957 3188	25/09/1989	
A1404	Black, Gillian	22 Moton Street, Marrick	2090	324 444	01/05/1994	
R A1509	Redford, Robert	122 Arthur Street, North	2060	9573188	19/02/1995	
①     ①     ②     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ③     ④     ③     ④     ④     ④     ④     ④     ④     ④     ④     ④     ④     ④     ④     ④     ④     ④     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑤     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑥     ⑦     ⑧     ⑧     ⑧     ⑥     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     ⑧     □     □     □     □     □     □     □     □     □     □	Purchasing	123 Pacific Highway,, N	2000	952 6475		
🗊 💼 03	Accounting	252 Canterbury Road,, C	2044	560 3633		
🕀 💼 04	Sales & Marketing	121 Pitt Town Road, Pitt	2345	364-8905		
🖻 💼 05	Maintenance	121 Railway Parade, W	2034	(02) 456-7896		
06	Personnel	121 Smith St, Newtown,	2067	367-4894		
09	Vehicle Maintenance	121 Smith Street, Newto	2015	(02) 562-2783		
AUD	Internal Auditing					
FLT	Fleet Administration					
GAC	Group Accounts					
INF	Information Services					
C LEG	Legal					
m MIS	Mamt. Information					
						>

To meet these objectives you will complete the following steps:

Step 1. Create Form iiiVFW11 - Tree View with ColumnsStep 2. Complete Form iiiVFW11 - Tree View with ColumnsStep 3. Add Pop-Up Menu to Show/Hide Columns - OptionalSummary

#### Step 1. Create Form iiiVFW11 - Tree View with Columns

- 1. Define a new field in the repository **iiiSTRDTE Start Date** based on **STD_OBJ**. This is a character field to be used as the date column in the tree view.
- 2. Create an initial version of your form.

#### Name: iiiVFW11

#### Description: Tree View with Columns.

Replace the form's code with the source from VFW062 - Appendix A.

Change references to **iiiSTRDTE** to use your initials.

Ignore other errors at this stage, you will add the missing code.

- 3. Review the form in the Design view and note the following:
  - The Tree View has a *ViewStyle* property of Unlevelled.
  - Most of the tree view columns are based on standard fields such as STD_OBJ rather than fields from the files which will be used to populate the columns. This is because the column data must be set up as required for each level.
  - All the columns are at Level 1.
  - *Drag Columns* is enabled. This means that the user will be able to drag and reposition the columns.
- 4. Review the source code that has been provided. If you are not familiar with a program, you should always use the *GoTo* tab to quickly understand what routines it includes.
  - When the *Add_Entry* method is invoked it is passed the variables to populate the tree level being filled.
  - After adding an entry to the tree view (*Name* = **Personnel**), the parent item is set for the new row in the line:

Set Com(#Personnel.currentitem) Parentitem(#i_Parent_item)

- An appropriate icon is added, depending on the level being populated.
- The **PB_LOAD** push button *Click* event in the supplied code simply clears the tree.

#### Step 2. Complete Form iiiVFW11 - Tree View with Columns

1. Create the Add_Departments method routine using the following code

Mthroutine Name(Add_Departments) Access(*private) Define_Com Class(#prim_tvit) Name(#Department_item) Reference(*dynamic Select Fields(#Deptment #deptdesc) From_File(Deptab) #std_code := DEP #depnull := *null #com_owner.Add_Entry I_Code(#deptment) I_Description(#deptdesc) O_Tree

* Add the sections for the department #com_owner.Add_Sections I_Parent_Item(#Department_item) I_Department(# Endselect Endroutine

Ignore the *Add_Sections is not* . . . error. You will add this routine in a later step.

- 2. Review the *Add_Departments* method routine:
- A dynamic reference to a tree view item component (#Department_item) is defined (#prim_tvit)
- The routine selects all records from table DEPTAB
  - A Group_By #DEPNULL is used to initialize fields in the tree
  - Invokes the Add_Entry method routine for each department
  - The Add_Entry is passed values for all tree columns
  - Add_Entry is also passed the type of level being added (I_Type)
  - The Add_Section method is invoked, to add all sections for each department.
- 3. Create an **Add_Sections** method routine using the following code:

Mthroutine Name(Add_Sections) Access(*private) Define_Map For(*input) Class(#prim_tvit) Name(#i_parent_item) Pass(*by_re Define_Map For(*input) Class(#deptment) Name(#i_department) Define_Com Class(#prim_tvit) Name(#Section_item) Reference(*dynamic) Select Fields(#section #secdesc #secaddr1 #secaddr2 #secaddr3 #secpCODE # #std_code := SEC

```
#iiiSTRDTE := *null
```

#std_textl := #secaddr1.trim + ', ' + #secaddr2.trim + ', ' + #secaddr3.trim
#com_owner.Add_Entry I_Code(#section) I_Description(#secdesc) O_Tree_It
* Add the Employees for the section

#com_owner.Add_Employees I_Parent_Item(#Section_item) I_Department(#d
Endselect

* Set a margin on the last item to help separate the groups of tree items #Personnel.Currentitem.marginbottom := 5 Endroutine

Ignore the *Add_Employees is not* . . . error, which you will correct later.

- 4. Review the *Add_Sections* method routine logic:
  - A dynamic reference to a tree view component (#Section_item) is defined.
  - Records are selected from the table SECTAB, for the received department code **i_department**.
  - The *Add_Entry* method routine is invoked for each section record retrieved.
  - The section level populates all columns, except Date, which is initialized as *Null.
  - The **i**_parent_item is passed by reference. This is the department tree view item.
  - For each section the *Add_Employee* method is invoked to add employees for each department / section.
  - A reference to the section tree view item is passed (#Section_item) to the *Add_Employees* method routine.
  - A *MarginBottom* property is set for current item, after employees for the section have been added.
- 5. Create an *Add_Employee* method routine based on the following code:

Mthroutine Name(Add_Employees) Access(*private) Define_Map For(*input) Class(#prim_tvit) Name(#i_parent_item) Pass(*by_re Define_Map For(*input) Class(#deptment) Name(#i_Department) Define_Map For(*input) Class(#section) Name(#i_Section)

Define_Com Class(#prim_tvit) Name(#Employee_item) Reference(*dynamic)
#std_code := EMP

Select Fields(#empno #givename #surname #address1 #address2 #address3 #p #std_textl := #address1.trim + ', ' + #address2 + ', ' + #address3 #iiiSTRDTE := #startdte.asdate( DDMMYY ).asdisplayString( DDsMMsCCY #com_owner.Add_Entry I_Code(#empno) I_Description(#Surname.trim + ', ' + Endselect

* Set a margin on the last item to help separate the groups of tree items #Personnel.Currentitem.marginbottom := 5 Endroutine

Use this code to create date field iiiSTRDTE if your date format is MMDDYY:

#iiiSTRDTE := #startdte.asdate( MMDDYY ).asdisplayString( DDsMMsCCY
 Note: Retrieving the real date field (STRDTER) and converting this using
 the following code, avoids the need for two versions of the code:

#iiiSTRDTE := #startdter.asdate( YYMMDD ).asdisplayString( DDsMMsCCY

- 6. Review the *Add_Employee* method routine logic:
- A dynamic reference to a tree view item *Employee_item* is defined.
- Records are selected from the logical file PSLMST1 using the passed department and section codes (*i_department* and *i_section*).
  - *Add_Employee* populates all tree view columns.
  - Start Date is shown in an alpha column, so that it can be set to blank for the department and section levels.
- 7. Complete the load push button click event. Your code should now look like the following. New code is highlighted in red, italic.

Evtroutine Handling(#pb_load.Click) Clr_List Named(#Personnel) #com_owner.Add_departments Endroutine

8. Review the *Add_Entry* method routine again.

#### Note:

• The routine has an output parameter #o_tree_item which is a reference to the current tree view item

- After each row is added (*add_entry to_list(#Personnel*) a reference to the current tree view item is obtained via:
   Set Com(#Personnel.currentitem) Parentitem(#i_Parent_item)
- This reference is returned to the calling method routine via:
   * Return the tree item for use as a parent
   Set_Ref Com(#o_Tree_item) To(#Personnel.currentitem)
- 8. Compile your form and test it.

## Step 3. Add Pop-Up Menu to Show/Hide Columns - Optional

1. In the *Design* view, drag and drop a *Pop-up Menu* onto the *Tree View* component

	Design Sou	urce   Multilingual Details	Repository Help   Cro
	ltem1		
		Code	Description
		Part ABCDEFGHIJ	aAbBcCdDeEfFgGhk
L		· · · · · · · · · · · · · · · · · · ·	m

The *Popup menu* will be displayed at the top of the *Design* panel.

**Hint:** Enter the menu item name and press *Enter* to define the next menu item.

2. Define the *Popup* menu items as:

Address

Zip Code

Bus. Phone

Start Date

- 3. Select each menu item and set its *Checked* property to **True**.
- 4. Add the following code to show and hide the tree view columns:

Evtroutine Handling(#MITM_1.Click) If (#MITM_1.checked = true) Set Com(#TVCL_3) Visible(false) Set Com(#MITM_1) Checked(false) Else Set Com(#TVCL_3) Visible(true) Set Com(#TVCL_3) Displayposition(3) Set Com(#MITM_1) Checked(true) Endif Endroutine

```
Evtroutine Handling(#MITM_2.Click)
If (#MITM 2.checked = true)
Set Com(#TVCL_4) Visible(false)
Set Com(#MITM 2) Checked(false)
Else
Set Com(#TVCL_4) Visible(true)
Set Com(#TVCL_4) Displayposition(4)
Set Com(#MITM_2) Checked(true)
Endif
Endroutine
Evtroutine Handling(#MITM_3.Click)
If (#MITM_3.checked = true)
Set Com(#TVCL_5) Visible(false)
Set Com(#MITM_3) Checked(false)
Else
Set Com(#TVCL_5) Visible(true)
Set Com(#TVCL_5) Displayposition(5)
Set Com(#MITM_3) Checked(true)
Endif
Endroutine
Evtroutine Handling(#MITM_4.Click)
If (#MITM_4.checked = true)
Set Com(#TVCL_6) Visible(false)
Set Com(#MITM_4) Checked(false)
Else
Set Com(#TVCL 6) Visible(true)
Set Com(#TVCL_6) Displayposition(6)
Set Com(#MITM_4) Checked(true)
Endif
Endroutine
```

**Note:** In the above code, when a column is made visible the *DisplayPosition* of that column is set back to its original value.

- 5. Compile and test your form. Using the right mouse menu on the tree view you should now be able to show or hide columns.
- 6. You have completed this exercise.

#### Summary

#### **Important Observations**

- You can see a very effective implementation of a tree view with columns in the *Repository* tab of the Visual LANSA Editor.
- The Tree View with columns provides powerful techniques for presenting information in a drill down format, but with the ability to display much more information.

## What You Should Know

• How to define and implement a tree view with columns.

## VFW062 - Appendix A

Source for the initial version of form iiiVFW11 - Tree View with Columns

Function Options(*DIRECT)

Begin_Com Role(*EXTENDS #PRIM_FORM) Clientheight(499) Clientwidth Define_Com Class(#PRIM_ATLM) Name(#ATLM_1)

Define_Com Class(#PRIM_PANL) Name(#PANL_1) Displayposition(1) Heig Define_Com Class(#PRIM_ATLI) Name(#ATLI_1) Attachment(Bottom) Man

* A new property called Viewstyle has been added to the PRIM_TRVW

* Unlevelled means that the tree no longer creates a level per field, and ins

* The developer is now reponsible for the level at which a tree item exists.

* This is governed by setting the PARENTITEM property of a tree item to ano

* NOTE - The tree can still be processed using SELECTLIST.

* Entries will be returned in the sequence they were added to the list. This may * To process in order sequence, use the FOR command

Define_Com Class(#PRIM_TRVW) Name(#Personnel) Columnbuttonheight(2 Define_Com Class(#PRIM_ATLI) Name(#ATLI_2) Attachment(Center) Mana Define_Com Class(#PRIM_TVCL) Name(#TVCL_1) Caption('Code') Captior Define_Com Class(#PRIM_TVCL) Name(#TVCL_2) Caption('Description') C

Define_Com Class(#PRIM_PHBN) Name(#pb_load) Caption('Load Tree') Dis Define_Com Class(#PRIM_TVCL) Name(#TVCL_3) Caption('Address') Capt Define_Com Class(#PRIM_TVCL) Name(#TVCL_4) Caption('Zip Code') Caption Define_Com Class(#PRIM_TVCL) Name(#TVCL_5) Caption('Bus. Phone') C Define_Com Class(#PRIM_TVCL) Name(#TVCL_6) Caption('Start Date') Ca *

Group_By Name(#allfields) Fields(#std_obj #std_desc #std_textl #postcode #s Group_By Name(#depnull) Fields(#std_textl #postcode #std_descs #iiiSTRDT Override Field(#postcode) Edit_Code(D)

Evtroutine Handling(#pb load.Click)

Clr_List Named(#Personnel)

Endroutine

Mthroutine Name(Add_entry) Access(*private)

Define_Map For(*input) Class(#std_obj) Name(#i_code)

Define_Map For(*input) Class(#std_Desc) Name(#i_Description)

Define_Map For(*input) Class(#prim_tvit) Name(#i_Parent_item) Mandatory(

```
Define_Map For(*output) Class(#prim_tvit) Name(#o_Tree_item) Mandatory(
```

```
Define_Map For(*input) Class(#std_code) Name(#i_type)
```

```
Define_Map For(*input) Class(#std_textl) Name(#i_addr)
```

Define_Map For(*input) Class(#postcode) Name(#i_zip)

Define_Map For(*input) Class(#std_descs) Name(#i_phone)

Define_Map For(*input) Class(#iiiSTRDTE) Name(#i_date)

#allfields := *null

* The same fields are used regardless of the "level" of the tree item

#std_obj := #i_code

```
#std_desc := #i_Description
```

#std_textl := #i_addr

#postcode := #i_zip

#std_descs := #i_phone

#iiiSTRDTE := #i_date

```
Add_Entry To_List(#Personnel)
```

* Set the new tree item's parent to the supplied parent. A null parent means that

* The parent is no longer governed by the data. It is a choice the developer can * Effectively, there is NO LIMIT to the number of levels.

* Note: The parent of a tree item is completely dynamic. It can be set at any tin Set Com(#Personnel.currentitem) Parentitem(#i_Parent_item)

* Set required image for each level

Case (#i_type)

When ('= DEP')

```
Set Com(#Personnel.currentItem) Image(#vi_deptcl)
```

When ('= SEC')

```
Set Com(#Personnel.currentItem) Image(#vi_sectcl)
```

When ('= EMP')

```
Set Com(#Personnel.currentItem) Image(#vi_employ)
```

Endcase

* Return the tree item for use as a parent

```
Set_Ref Com(#o_Tree_item) To(#Personnel.currentitem)
```

Endroutine

End_Com

# VFW070 – Create a Reusable Part Object

The reusable parts you have built in previous exercises have been visual components. A reusable part of type Panel has an ancestor of PRIM_PANL.

A reusable part of type Object is non-visual, and has an ancestor of PRIM_OBJT. Think of this example as a component which you call (invoke) to perform some processing and return a result.

# Objective

This exercise will build a simple reusable object which will calculate the years, months and days since an employee joined the company.

The component is invoked when passed the start date and it returns the three result values.

- To build and implement a 'Time Employed' reusable part
- To learn how to write a method routine, which has input and output parameters
- To use the new component in the *Employee Details* command handler

To achieve these objectives you will complete the following:

Step 1. Create Time Employed Reusable Part

Step 2. Implement Time Employed calculation in Employee Details Command Handler

Summary

# **Before You Begin**

You should complete exercises VFW030, VFW040 and VFW042.

#### **Step 1. Create Time Employed Reusable Part**

- Create a new *Reusable Part /* type *Object*: *Name*: iiiVFW12 Description: Time Employed Calculator
- 2. Note that your reusable part has an ancestor of **PRIM_OBJT**.
- 3. iiiVFW12 should have one method routine, *uEmployTime*, which has one input parameter and three output parameters, defined as follows.

ForNameClass*inputi_strdterSTARTDTER*outputo_yearPRIM_NMBR*outputo_monthPRIM_NMBR*outputo_daysPRIM_NMBR

**Note:** STARTDTER will be passed to this component, which is a 6 long date in the form YYMMDD. This will provide a solution which works for regions using both DDMMYY and MMDDYY date formats.

Your *uEmployTime* method routine should initially look like the following:

Mthroutine Name(uEmployTime) Define_Map For(*input) Class(#startdter) Name(#i_strdter) Define_Map For(*output) Class(#PRIM_NMBR) Name(#o_year) Define_Map For(*output) Class(#prim_nmbr) Name(#o_month) Define_Map For(*output) Class(#prim_nmbr) Name(#o_days) Endroutine

4. Define_Map Statements:

This step reviews the Define_Map statement, there is no code for you to write here.

The *Define_Map* statement defines input and output parameters to a routine.

For example:

Define_Map For(*input) Class(#startdter) Name(#i_strdte)

- The method routine will reference this variable as **#i_strdte**.
- This method will be invoked by another components, passing the required parameters and values:

Invoke #iiiVFW10.uEmployTime i_strdte(#startdter) . . .

- The invoking component refers to this parameter as **i_strdte()**.
- Input parameters are mandatory unless they are defined with a default value, for example, this is an optional input parameter:

Define_Map For(*input) Class(#prim_nmbr) Name(#i_num) Mandatory(1)

**Note:** Start Date is a signed 6.0 field. Designing this component to accept only this form of input is obviously a limitation. In a real example, you would probably add another input parameter which defines the format of the passed date, and also change the input date definitions so that a number of date formats of such as Signed 6,0 and Signed 8,0 and Date or Datetime can be handled.

5. Define the following work fields (following the Begin_Com) which are required during the calculation routine:

Define Field(#i_date) Reffld(#std_datex) Desc('Input Date') Define Field(#c_date) Reffld(#std_datex) Desc('Current Date') Define Field(#i_year) Reffld(#yyyy) Desc('Input Year number') Define Field(#c_year) Reffld(#yyyy) Desc('Current Year number') Define Field(#i_month) Reffld(#month) Desc('Input Month number') Define Field(#c_month) Reffld(#month) Desc('Current Month number') Define Field(#i_day) Reffld(#day) Desc('Input Day Number') Define Field(#c_day) Reffld(#day) Desc('Output day number')

- 6. In the *uEmployTime* method, start to build the logic to calculate years, months and days since the employee start date:
- 7. Setup current and input year, month and day values:

```
* Setup current and input year, month and day values.
#c_date := *date
```

```
#c_year := #c_date.year
#i_date := #i_strdter.asdate( yymmdd )
#i_year := #i_date.year
#i_month := #i_date.month
#i_day := #i_date.day
```

These calculations make use of intrinsic functions. Use *F2 Features* help on a field to discover the intrinsic functions which it supports.

8. Calculate number of years

Current year must be greater than or equal to the input year

```
* Calculate number of years, months and days
If (#c_year >= #i_year)
#o_year := #C_year - #I_year
Else
#o_year #o_month #o_days := 0
Endif
```

9. Extend the logic to calculate number of months. New code is highlighted in red

```
If (#c_year >= #i_year)
#o_year := #C_year - #I_year − 1
* Calculate number of months
* When input month is less than current month
If (#i_month < #c_month)</pre>
#o_month := #c_month - #i_month - 1
Else
#o_month := (12 - #i_month) + #c_month - 1
#o year -= 1
Endif
If (#i month = #c month)
#o month := *zeroes
Endif
Else
#o_year #o_month #o_days := 0
Endif
```

10. Extend logic to calculate number of days. New code is highlighted in red.

```
If (#c_year >= #i_year)
#o_year := #C_year - #I_year - 1
* Calculate number of months
* When input month is less than current month
If (#i_month < #c_month)</pre>
#o month := #c month - #i month - 1
Else
#o_month := (12 - #i_month) + #c_month - 1
#o_year -= 1
Endif
If (#i_month = #c_month)
#o month := *zeroes
Endif
* Calculate number of days
* when input day number is less than current day
If (#i_day < #c_day)
#o_days := #c_day - #i_day
#o month += 1
Else
If (#i_month = 2)
#o_days := (28 - #i_day) + #c_day
Endif
If ((#i_month = 4) *Or (#i_month = 6) *Or (#i_month = 9) *Or (#i_month
#o_days := (30 - #i_day) + #c_day
Else
#o_days := (31 - #i_day) + #c_day
Endif
Endif
Else
#o year #o month #o days := 0
Endif
```

```
11. Compile your Time Employed reusable part.
```

#### **Step 2. Implement Time Employed calculation in Employee Details Command Handler**

- 1. Open the *Employee Details* command handler in the editor (iiiVFW06).
- 2. Add a Group Box component to the main panel (BODY_HEAD) and change its *Caption* to **Time Employed**.
- 3. Add field STD_NUM to the group box. Set up the followings properties:

Property	Value	
Caption	Years	
LabelType	Caption	
Name	EMP_YEARS	
MarginLeft	: 50	

Reduce the field's width as necessary.

- 4. Add another STD_NUM field to the group box and repeat step 3 the above steps except:
  - a. Change *Name* to **EMP_MONTHS**
  - b. Change *Caption* to **Months**
- 5. Add another STD_NUM field to the group box and repeat step 3, except:
  - a. Change *Name* to **EMP_DAYS**
  - b. Change *Caption* to **Days**
- 6. Adjust the width of each field as required.
- 7. On the *Design* ribbon, use *Edit / Align* button to arrange the fields.



8. Save your changes.

Your design should look like the following:

. . . . . . . . . . Termination Date (DDMMYY) ... 123456 Time Employed Years 1,234,567 Month 1,234,567 . . . . . . . Years 1,234,567

- 9. In the *Design* view, drag and drop your *Time Employed* reusable part onto *Employee Details* component. This will create a *Define_Com* for iiiVFW12. Change its *Name* to EMPLOY_CALC
- 10.Change the Group_By XG_HEAD to include STARTDTER.
- 11. In the *uExecute* method routine add the following code, following the FETCH command:

If_Status Is(*okay) Invoke Method(#EMPLOY_CALC.uEmployTime) I_Strdte(#STARTDTER) C Endif

12. In the SAVE_BUTTON *Click* event routine, add the following logic, after the UPDATE command:

If_Status Is(*OKAY) Invoke Method(#EMPLOY_CALC.uEmployTime) I_Strdte(#STARTDTER) C Endif

- 13. Compile the *Employee Details* command handler.
- 14. Execute the *Framework as an End User* and test the *Time Employed* calculator.
- 15. Change *Start Date* for an employee to a recent date, for example, 2012 or 2013, to easily check the results are as expected. Note, this is a 6 digit date.

#### Summary

#### **Important Observations**

- Reusable parts based on PRIM_OBJT are non-visual components. They can be used to create callable (invoke) modules which can be shared by your Windows and web applications (WAMs).
- This type of component can also be created to store application settings and values which are shared across many components in the application system.

## **Tips & Techniques**

• When creating this type of component, ensure that its logic will be common and re-used. Otherwise there won't be any benefit from making it a separate component.

## What You Should Know

• How to implement a reusable part to perform calculations.

## VFW072 – Create a Department Dropdown Reusable Part

In an earlier exercise, you added a *Combo box* for departments to the Employee Details component, by adding it directly into the logic of the command handler. Clearly this approach would not be the best one from a maintenance point of view, and does not contribute to simplifying or re-using code.

A reusable part provides a component which can be used to create common application components, such as a department's dropdown list, which can be developed once, and then widely re-used throughout the application.

#### Objective

- To create a reusable part (RP)which manages a combo box (or dropdown) containing a list of all valid departments.
- To populate the RP combo box from the table DEPTAB.
  - The 'Department Dropdown' RP will publish a property which enable its current department code to be set, or for another component to get its current value.
  - When the current department property is set, the Department Dropdown RP will position the combo box to the new value.
  - The 'Department Dropdown' reusable part will publish an event, which it will signal when the selected department has changes.

🎒 Employee : Details (A10	005-SMITHS PETER)
Details 😡 Brief Notes	🎐 Image 💊 Notes 👙 Skills 2 👙 Skills
Employee Number	A1005 Save
Employee Surname	SMITHS
Employee Given Name(s)	PETER
Street No and Name	72 Mullane Avenue,
Suburb or Town	BAULKHAM HELLS.
State and Country	NSW. Department Dropdown
Post / Zip Code	2147 IIIVEVV13
Home Phone Number	674 4316
Business Phone Number	777 7265
Department	INTERNAL AUDITING
Section Code	02
Employee Salary	46,700.04
Start Date (DDMMYY)	1/02/71
Termination Date (DDMMYY)	0/00/00
Time Employed	
- Yeses many	me and a manufacture

To meet these objectives, you will complete the following:

Step 1. Create Department Dropdown Reusable Part

Step 2. Make the Reusable Part Useful

Step 3. Add Department Dropdown to Employee Details command handler

Step 4. Complete Command Handler to use Department Dropdown Summary

# **Before You Begin**

Complete exercises VFW04, VFW05, VFW06.

#### Step 1. Create Department Dropdown Reusable Part

1. Create a new Reusable Part / Panel:

#### Name: iiiVFW13

#### Description: Department Dropdown

2. Resize the *Panel* so that it looks like the following:



- a. Add a *Combo box* to the panel.
- b. Select the *Details* tab and change the *ComboBoxStyle* property to **DropDownList**.
- c. Drop fields DEPTDESC and DEPTMENT into the *Combo box*. If you add them in this order, you have created CBCL_1 sourced from DEPTDESC and CBCL_2 sourced from DEPTMENT.
- d. On the *Details* tab, ensure the column sourced from DEPTMENT is selected. You can select any component within this reusable part from this dropdown.

Details	
CBCL_2	- U
Properties Events Methods	
<b>•</b> ?	
ComponentClassNam	PRIM_CBCL
ComponentPatternNa	PRIM_CBCL
ComponentTag	
ComponentTypeNam	PRIM CBCL

- e. Change its *Visible* property to **False**. Only the column sourced from DEPTDESC should be visible in the combo box.
- f. Resize the *Combo box* so that it can display the full description.
- g. Move the *Combo box* to the right and add a *Label* component onto the left side of the *Panel*. Give the *Label* a *Caption* of **Department**.
- h. Resize the *Label*, to fit the caption text and move it to the top left of the *Panel*.

Hint: An effective way to position a component in some cases, is to set its *Left* and *Top* properties. In this case, setting these to **0**, will achieve the required result.

- i. Set *Combo box Left* property to **150**, and the *Top* property to **0**. This will make the department dropdown component suitable for using alongside fields on a panel which is managed by a flow down manager.
- j. Resize the panel so that it uses the minimum space.

Your design should now look like the following:



The reusable part will occupy the space defined by its *Panel*, when you drop it onto your application form. It is important to minimize the space used by the reusable part, using the techniques shown above.

**Note:** You could also make this component more flexible by coding it to optionally hide the label and, if required, resize to display only the combo box.

3. Create an *Initialize* event handling routine for the combo box. The logic in this routine should fill the combo box from the table DEPTAB and ensure that the focus is set to the first entry (if there was one). You code should look like the following:

```
Evtroutine Handling(#CMBX_1.Initialize) Options(*NOCLEARMESSAGES
Clr_List Named(#CMBX_1)
Select Fields(#CMBX_1) From_File(deptab)
Add_Entry To_List(#CMBX_1)
Endselect
Get_Entry Number(1) From_List(#CMBX_1)
If_Status Is(*okay)
#CMBX_1.currentItem.focus := true
Endif
Endroutine
```

If you compiled your reusable part now, and added it to your command handler, it would contain data for all departments. However, it is not yet capable of interacting with your command handler.

## Step 2. Make the Reusable Part Useful

In this step you will extend your reusable part, so that it can interact with another reusable part or form which uses it.

- To do this you will add/publish a new event which will be signaled when the department selected in the *Combo box* changes.
- You will also add/publish a property based on DEPTMENT. (That is, with *Class(#Department)*. When this property is set, it will execute a property routine to re-position the *Combo box*. When another component "gets" the property, the Department Dropdown reusable part will return the current value of DEPTMENT .
- 1. Define an event, named *uDepartmentChanged*.

Define_Evt Name(uDepartmentChanged)

User defined events, properties and methods in these exercises have a naming policy starting with **u**. You may want to implement some similar policy, which makes it easy for other developers to quickly identify the features that they may be interested in using.

2. Create an *ItemGotFocus* event handling routine for the *Combo box*. Add logic to signal the *uDepartmentChanged* event.

Also add code to signal this event at the end of the *Combo box Initialize* event. When this component is used in a "New Employee" command handler for example, the application will be able to initialize its department code, by handling this event.

Evtroutine Handling(#CMBX_1.ItemGotFocus) Options(*NOCLEARMESSA Signal Event(uDepartmentChanged) Endroutine

3. Define a property *uCurrDepartment*. It should handle Get, by returning the current value of DEPTMENT. It should handle Set, by executing a *SetDept* property routine.

Define_Pty Name(uCurrDepartment) Get(*auto #deptment) Set(SetDept)

**Note:** the editor will highlight this code as having an error, until you have

created the property routine.

- 4. Create a property routine, named *SetDept*.
  - a. Define a map, for input, with a class of DEPTMENT, named #SetDept. A property routine must have one input or output parameter. In this case the routine receives DEPTMENT as SetDept and will reposition the combo box.

Your code should look like the following:

Ptyroutine Name(SetDept) Define_Map For(*input) Class(#deptment) Name(#SetDept) Endroutine

b. Add logic to read through the combo box entries, using SELECTLIST/ENDSELECT.

Leave when DEPTMENT equals SETDEPT.

Then set focus for the combo box current item.

Your code should now look like:

```
Ptyroutine Name(SetDept)
Define_Map For(*input) Class(#deptment) Name(#setdept)
Selectlist Named(#CMBX_1)
Leave If(#deptment = #setdept)
Endselect
#CMBX_1.currentItem.focus := true
Endroutine
```

5. Compile your department dropdown reusable part.

# **Step 3. Add Department Dropdown to Employee Details command handler**

In this step you will change your Employee Details command handler to use your new reusable part.

- 1. Open the *Employee Details* command handler iiiVFW06 in the editor.
- 2. Delete the department combo box which you added in VFW050.
  - a. Switch to the source and locate the errors which this action has created.
  - b. Delete the *column definitions* for the combo box (for example, CBCL_1) and the *flow manager item* (for example, FWLI_4).
  - c. Delete all the code associated with the combo box from the uExecute and uInitialize method routines
  - d. Compile your component to check that you have no errors.
- 3. Drop the Department Dropdown RP (iiiVFW13) onto the command handler left hand panel (BODY_HEAD). Because this panel has a flow down manager, your component will be positioned below the existing fields.
- 4. Position the department dropdown:
  - a. Select the existing department field and check its DisplayPosition which should be 10.
  - b. Give your department dropdown component a DisplayPosition property of 10.
- 5. Delete the existing department code field from the panel.

# **Step 4. Complete Command Handler to use Department Dropdown**

In this step you will add code to the *Employee Details* command handler, to interact with the *Department Dropdown* component.

Your command handler is a maintenance function. This means:

1. When the employee data is read, it needs to tell the department dropdown to position to the current employee's department. Change the *uExecute* method routine to set the *uCurrDepartment* in the department dropdown component to **DEPTMENT**.

Your code should look like the following. Changes are shown in red.

```
• • • •
```

Fetch Fields(#XG_HEAD) From_File(PSLMST) With_Key(#EMPNO)
* ??? Addition logic may be required here ???
#IIIVFW13.ucurrDepartment := #deptment

- 2. When the department dropdown component signals that department has changed, the command handler should get the current department code value from the department dropdown.
- 3. Add an event handling routine for *iiiVFW13.uCurrDepartment*. Your code should look like the following:

```
Evtroutine Handling(#IIIVFW13.UDepartmentChanged)
#deptment := #IIIVFW13.uCurrDepartment
Endroutine
```

**Hint:** To create the event handling routine, on the *Design* tab, select the department dropdown and use the context menu to select *Events: IIIVFW13 / uDepartmentChanged Event*.

- 4. Compile the Employee Details command handler, iiiVFW06.
- 5. Execute the *Framework as an End User*. Use the By Location filter to populate the instance list and check that the department dropdown displays the correct value.
- 6. Change the department for an employee and *Save* the change. You may find that the change is invalid because the section code is no longer valid. You will address this issue in the next exercise.

## **Step 5. Document your Event and Property**

These exercises have emphasized how to discover the help text build into all shipped components by using F2 *Features* help. In this step you will learn how to document your own components.

When designing and building your application, reusable parts enabled you to create components designed for re-use, such as the department dropdown which you have just built. These components will be much more useful if you document them so that other developers will easily understand them and want to use them.

1. Events, property and method statements have optional *Desc()* and *Help()* keywords. Help should be used to document your own reusable parts. *F2 Features help* will then be able to display this information for other developers.

Method routines will usually have one or more *Define_Map* statements. These should also be documented using *Help()*.

2. Open your department dropdown RP in the editor. Add appropriate help text to the event and property statements. For example:

Define_Evt Name(UDepartmentChanged) Help('This Event is signalled when | Define_Pty Name(uCurrDepartment) Get(*auto #deptment) Set(SetDept) Help

- 3. Recompile your component.
- 4. Switch to the Employee Details command handler in the editor (iiiVFW06). Select the department dropdown component in the *Design* view.
  - a. Either press *F2*, or use the context menu to select: *Reusable Part: IIIVFW11 / Features*.
  - b. Expand Properties and double click to display help for *uCurrDepartment*.
  - c. Expand the Events and double click on *uDepartmentChanged* to display its help.

#### Summary

#### **Important Information**

- This exercise created a very simple dropdown reusable part. The next exercise provides an example of a more functional reusable part, supporting two dropdowns.
- Build common reusable parts whenever it is clear that this functionality will be required in a number of places.
- Your reusable parts will be much more useful if you include help text for other developers.

## **Tips & Techniques**

• Adding a HideLabel property to the reusable part, together with property routines to set Margin Left to 0, or to the default 150, would make this dropdown more useful.

## What You Should Know

- How to create and design a visual reusable part.
- How to implement your own reusable part in an update command handler.

# VFW074 – Create a Compound Reusable Part

## Objective

As observed in testing, the department dropdown component, from an end user point of view has limited benefits, because if department is changed, the available valid section codes changes.

- This exercise builds a Departments and Sections dropdown reusable part.
- The new component will re-use the department dropdown component and manage a sections dropdown, based on the current department.

🎒 Employee : Details (A0193-SMITHSON FRED) 📃			
Details 🛛 🚱 Brief Notes	🥵 Image   💊 Notes   👶 Skills 2   🁙 Skills		
Employee Number	A0193 Save		
Employee Surname	SMITHSON Compound Reusable Part		
Employee Given Name(s)	FRED		
Street No and Name	121 Cutter Ave Department / Section		
Suburb or Town	Windsor Dropdown		
State and Country	NSW iiiVFW14		
Post / Zip Code	2034		
Home Phone Number	(02) 546-4657		
Business Phone Number	(02) 354-5647		
Department	ADMINISTRATOR DEPT		
Sections	INTERNAL ADMIN SRV		
Employee Salary	35,000.04		
Star Date (DDM-40)	3/07/00		

To achieve these objectives you will complete the following:

Step 1. Create Department / Section Dropdown Reusable

Step 2. Make the Department / Section Dropdown Useful

Step 3. Modify Department Dropdown

Step 4. Implement the Compound Reusable Part Summary

## **Before You Begin**

Complete exercises VFW030, VFW040, VFW042 and VFW072.
### Step 1. Create Department / Section Dropdown Reusable

1. Create a new reusable part:

#### Name: iiiVFW14

#### Description: Department / Section Dropdown

- 2. Resize the panel and drop the department dropdown reusable (iiiVFW13) onto the panel.
  - a. Change its *Name* to **DEPT_DD**
  - b. Give DEPT_DD a *Top* and *Left* property of **0**.

Your design should look like the following:

	11	1	/F	=\	N	1	4		-	D	e	p	a	r	tr	m	e	n	t	/	5	Se	ec	t	ic	or	ı	D	r	0	p	d	C	v	vr	n	l	*	
De	eķ	pạ	ņ	ņ	ņē	ņ	ť	:	:	:	:	:	:	:	:	:	:	[	a/	٩ł	bE	8c	C	d	D	eł	f	Fg	g(	Gł	h	li	ij.	J				Ŧ	٦
•										•	•	•			•	•	•	÷	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	÷
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	·	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•	•	•		•	•	•	•			•	•			•			•		•	•	•	•		•	•		•		•		•

- 3. Drag and drop a combo box onto the panel.
  - a. Change its *Name* to **SECT_DD**.
  - b. Find the Table SECTAB on the Repository tab.
  - c. Drop fields SECDESC and SECTION into the combo box.
  - d. Change the column sourced from SECTION *Visible* property to **false**. The combo box should be displaying the section description only. Change the *ComboBoxStyle* to **DropDownList**.

A **DropDownList** style does not allow the user to enter a value. Only one of the displayed values may be selected.

- e. Resize the Combo box so that all of the section description text is visible.
- f. Position the sections Combo box closely underneath the department Combo box.
- g. Drop a Label component to the left of the sections Combo box and changes its *Caption* property to **Sections**. Adjust the *Width* and *Height* of the Label.
- h. Move the Label so that it is close underneath the "Departments" label and change the new Label's *Left* property to **0**.

**Hint:** To move components around accurately in the *Design* view, use the *Ctrl+Cursor* keys.

i. Change the SECT_DD Combo box's *Left* property to **150**.

Your design should look like the following:

IIIVFW14 - Departme	nt / Section Dropdown	x
Department ::::::::::::	aAbBcCdDeEfFgGhHiljJ	*
Sections	aAbBcCdDeEfFgGhHiljJ	+

4. Save your changes.

# Step 2. Make the Department / Section Dropdown Useful

From the previous exercise, you know that the Departments dropdown component:

- Signals *uDepartmentChanged* when the selected department is changed.
- Has a *uCurrDepartment* property which re-positions the dropdown or returns the currently selected department code.
- 1. Consider what functionality the new department / section component will need, to interact with your *Employee Details* command handler:
  - The first thing to recognize is that the new component is now responsible for interacting with the department dropdown component. The department dropdown component is encapsulated within the new reusable part. This means the new reusable part must handle both department and section changes.
  - For example, if its current department property is changed it must:
  - a. Reposition the department dropdown
  - b. Rebuild the sections dropdown from the table SECTAB using the new department code as a key.
- 2. In the *Design* view, select the department dropdown RP and create an event handling routine for *uCurrDepartmentChanged*. Add logic to do the following:
  - a. Set DEPTMENT to the value of the *uCurrDepartment* property of DEPT_DD.
  - b. Signal that department has changed
  - c. Clear the list SECT_DD
  - d. Select SECTION and SECDESC from file SECTAB using DEPTMENT as key, add all entries to SECT_DD
  - e. Get entry 1 from SECT_DD
  - f. For SECT_DD, set *Focus* to current item

Your code should look like the following:

Evtroutine Handling(#DEPT_DD.UDepartmentChanged) #deptment := #DEPT_DD.uCurrDepartment Signal Event(uDeptChanged) Clr_List Named(#SECT_DD) Select Fields(#SECT_DD) From_File(sectab) With_Key(#deptment) Add_Entry To_List(#SECT_DD) Endselect Get_Entry Number(1) From_List(#SECT_DD) #SECT_DD.currentItem.focus := true Endroutine

3. Define the following properties:

Define_Pty Name(uCurrDept) Get(GetCurrDept) Set(SetCurrDept) Define_Pty Name(uCurrSection) Get(*auto #section) Set(SetCurrSection)

Setting or getting *uCurrDept* will require logic in property routines.

Getting *uCurrSection* will return current value of SECTION.

Setting *uCurrSection* will require logic in a property routine.

Ignore errors at this stage.

4. Define the following events:

Define_Evt Name(uSectChanged)
Define_Evt Name(uDeptChanged)

5. Create the *GetCurrDept* property routine. This needs to an output parameter based on DEPTMENT.

Your code should look like the following:

Ptyroutine Name(GetCurrDept) Define_Map For(*output) Class(#deptment) Name(#CurrDept) Endroutine

6. The *GetCurrDept* routine needs to retrieve the current department from DEPT_DD. Add code to do this. The new code is shown in red.

Ptyroutine Name(GetCurrDept) Define_Map For(*output) Class(#deptment) Name(#CurrDept) #CurrDept := #DEPT_DD.uCurrDepartment Endroutine

7. Create a *SetCurrDept* property routine. It needs an input parameter based on

DEPTMENT.

Your code should look like the following:

Ptyroutine Name(SetCurrDept) Define_Map For(*input) Class(#deptment) Name(#CurrDept) Endroutine

8. The *SetCurrDept* needs to set the current department property in DEPT_DD. Add code to do this. The new code is in red.

Ptyroutine Name(SetCurrDept) Define_Map For(*input) Class(#deptment) Name(#CurrDept) #DEPT_DD.uCurrDepartment := #CurrDept Endroutine

9. Create a *SetCurrSection* property routine. This needs to have an input parameter based on SECTION.

Your code should look like the following:

Ptyroutine Name(SetCurrSection) Define_Map For(*input) Class(#section) Name(#CurrSect) Endroutine

10. The *SetCurrSection* routine needs to re-position the section combo box to the input value.

It does this by reading the combo box using SELECTLIST and leaving when SECTION = the input section code.

It should then set focus for current item and signal the *uSectChanged* event.

Add code to achieve this. New code is shown in red.

Ptyroutine Name(SetCurrSection) Define_Map For(*input) Class(#section) Name(#CurrSect) Selectlist Named(#SECT_DD) Leave If(#section *EQ #CurrSect) Endselect #SECT_DD.currentItem.focus := true Signal Event(uSectChanged) Endroutine 11. Create an *ItemGotFocus* for the sections combo box. Add logic to signal the *uSectChanged* event.

Your code should look like the following:

Evtroutine Handling(#SECT_DD.ItemGotFocus) Options(*NOCLEARMESS/ Signal Event(uSectChanged) Endroutine

12. Compile your department / section combo box component.

## **Step 3. Modify Department Dropdown**

1. A change is required to iiiVFW13 to signal *uDepartmentChanged* when the departments dropdown is *Initialized*. The existing routine is as follows:

Evtroutine Handling(#CMBX_1.Initialize) Options(*NOCLEARMESSAGES Clr_List Named(#CMBX_1) Select Fields(#CMBX_1) From_File(deptab) Add_Entry To_List(#CMBX_1) Endselect Get_Entry Number(1) From_List(#CMBX_1) If_Status Is(*okay) #CMBX_1.currentItem.focus := true Endif Endroutine

Although the program is setting *Focus* for the first entry, this will not signal an *ItemGotFocus* event.

2. Add code to signal the *uDepartmentChanged* event, as follows. New code is shown in red, italics.

Evtroutine Handling(#CMBX_1.Initialize) Options(*NOCLEARMESSAGES Clr_List Named(#CMBX_1) Select Fields(#CMBX_1) From_File(deptab) Add_Entry To_List(#CMBX_1) Endselect Get_Entry Number(1) From_List(#CMBX_1) If_Status Is(*okay) #CMBX_1.currentItem.focus := true **Signal Event(uDepartmentChanged)** Endif Endroutine

3. Compile iiiVFW13.

# Step 4. Implement the Compound Reusable Part

In this step you will change your Employee Details command handler to use the new department / section combo box component.

- 1. Open the Employee Handler (iiiVFW06) in the editor
- 2. From the *Design* view, delete the department dropdown component (iiiVFW13) and the Department and Section fields
- 3. Switch to *Source* and remove all code which refers to iiiVFW13. You will find that the flow manager items for iiiVFW13, DEPTMENT and SECTION have also been removed, so no further corrections are needed for now.
- 4. Drag and drop your department / sections dropdown component onto the main left hand panel (BODY_HEAD).
  - a. Change its *Name* to **DEPSEC_DD**.
  - b. Change its *DisplayPosition* and *TabPosition* to **10**.
- 5. Select the DEPSEC_DD component and use the *Details / Events* tab to create a *uDeptChanged* and *uSectChanged* event handling routine.
- 6. Switch to the *Source* view. In the *uExecute* method routine add code to set the current department and section properties in DEPSEC_DD, after the employee data has been read. Your code should look like the following: New code is shown in red.

. . . . .

Fetch Fields(#XG_HEAD) From_File(PSLMST) With_Key(#EMPNO)
#DEPSEC_DD.uCurrDept := #deptment
#DEPSEC_DD.uCurrSection := #Section

The department / section component will position the department dropdown, rebuild the sections dropdown for this department code and position the sections dropdown.

8. Complete the logic for the department and section changed events from DEPSEC_DD.

In each case they simply need to set the command handler field to the related property from DEPSEC_DD. Your code should look like the following:

Evtroutine Handling(#DEPSEC_DD.uDeptChanged)
#deptment := #DEPSEC_DD.uCurrDept

Endroutine Evtroutine Handling(#DEPSEC_DD.uSectChanged) #section := #DEPSEC_DD.ucurrsection Endroutine

- 9. Compile your command handler.
- 10. Execute your Framework as an end user. Test the *Employee Details* command handler.
  - a. Ensure that the correct department and sections are displayed when an employee is displayed.
  - b. Select new department and section values in the dropdowns. Save the change and ensure the file is updated successfully.
  - c. Check that the sections dropdown is re-populated every time a new department is selected in its dropdown.

### Summary

### **Important Observations**

- This exercise provides a good example of designing a useful reusable part.
- Components like this one will probably be required in most applications.

# **Tips Techniques**

- Note how the Department / Section dropdown component encapsulates the department dropdown list and must provide an interface for both departments and sections.
- Remember, it is important to document your reusable parts to promote their re-use by other developers.

# What You Should Know

• How to build a simple compound reusable part.

# VFW080 – Using an Explorer Component

The *Explorer* component is used to view files and directories either on local hard disks or across the network.

You can use two *Explorer* components, one for showing directories and paths and one for showing the files in a folder (similar to Windows Explorer). Implement communication between the two components by using the *NotifyComponent* property of the first Explorer component.

# Objectives

- To create a command handler that displays an employee image.
- To create a *Find Employee Image* form that uses the *Explorer* components to enable the user to find and select an employee image.



To achieve the objectives you will complete the following:

- Step 1. Create Employee Images File
- Step 2. Create the Find Employee Image Form
- Step 3. Make the Find Image Form Useful
- Step 4. Create the Employee Image Command Handler
- Step 5. Complete the Image Command Handler

Step 6. Plug In and Test the Image Command Handler Summary

# **Before You Begin**

Complete exercises VFW030, VFW040 and VFW042.

# **Step 1. Create Employee Images File**

Create a new field:
 *Name:* iiiEMPIMG
 *Description:* Employee Image
 *Field type:* BLOB

You do not need to open the field in the editor.

 Create new *iiiEmpImages*.
 Make the file RDMLX enabled: *Name*: *iiiEmpImages Description*: *Employee Images*

Field Key

EMPNO 1

iiiEMPIMG

3. Compile the file. Ensure that your compile options build *Rebuild Tables* and *Rebuild OAMs*.

## Step 2. Create the Find Employee Image Form

1. Create a new / *Basic Form*:

Name: iiiVFW15

#### Description: Find Image

- 2. With the *Design* view selected, use the *Design* ribbon to add *New Layout / Attachment* manager to the form.
- 3. Drop a Panel onto the right hand side, and change its *Name* to **IMAGE_PANL**. Change its *Width* to approximately **125**.
- 4. Drop another Panel onto the centre of the form and change its *Name* to **SEARCH_PANL**.

If necessary, with the form selected, open the *Layout Helper* tab and use the *Child Details* tab to ensure the IMAGE_PANL is attached to the **Right**. SEARCH_PANL should be attached to the **Center**.

Layout Helper		
Layout Managed Component		٤
IIIVFW15		
Layout		🗳 🗙
ATLM_1		•
Children 🌒 🗙 🛯 🗙	Child Details As Child Details	
SEARCH_PANL	Category Attachment	•
	Top Left Bottom Right	Center None

- 5. As you are designing, save the component regularly.
- 6. Select the IMAGE_PANL, and use the *Design* ribbon to add a *Flow down manager*.
  - a. Add two *Push buttons* to the IMAGE_PANL.
  - b. Add a *Group Box* and resize it as shown. Give the *Group Box* a *Caption* of **Selected Image**.
  - c. Drop an *Image* component into the *Group Box* and resize the *Image* to fill the *Group Box* as shown:



d. Select the IMAGE_PANL. On the *Layout Helper* tab select *Layout Manager Details tab* tab. Select *Margins / Category*. Set the *Left* margin to **15** and the *Top* margin to **12**.

Layout Managed Component	6
IMAGE_PANL	
Layout	岱 )
FWLM_1	
Children 🔋 🗙 👔 🗙	Layout Manager Details Child Details As Child
V GPBX_1 V PHBN_1 V PHBN_2	Category Margins *
	All 0    Clear
	Left 15 ≎ Right 0 ≎

- 7. Set up the *Push Buttons*:
  - a. Set up the first push button properties as follows:

Property	Value
Caption	OK

# Name **PHBN_OK**

- b. Create a Click event for PHBN_OK.
- c. Set up the second Push button as follows:

Property	Value
Caption	Cancel
Name	PHBN_CANCL

- d. Create a Click event for PHBN_CANCL
- 8. Select the SEARCH_PANL and use the *Design* ribbon, to add a *Split Vertical* manager.

On the *Layout Helper tab*, *select the Layout Manager Details* tab, select the *Category / Divider Style* and select the **Raised** option.

- 9. Change the left hand Panel's *Name* to **FOLDERS**.
- 10. Change the right hand Panel's *Name* to **FILES**.
- 11. Select the FOLDERS panel. On the *Layout Helper* tab, add an *Attachment manager* by selecting ATLM_1 in the *Layout* drop down.
- 12. On the *Controls* tab, select *All Controls* and add an *Explorer* component into the center of the FOLDERS panel.

If necessary, use the *Layout Helper* tab / *Child Details* tab to ensure that it is attached in the **Center**.

Your design should look like the following:



13. Select the FILES panel and on the *Layout* tab, give it the ATLM_1 *Attachment manager*, by selecting it in the *Layouts drop down*.

Drop an *Explorer* component into the center of the FILES panel. With the FILES panel selected, on the *Layout Helper* tab, use the *Child Details* tab to check the explorer (DCBX_2) is attached in the **Center**.

- 14. Save your form design.
- 15. Select the left hand *Explorer* component (DCBX_1) and use the *Details* tab to set up its properties:

Property	Value
Display Style	DirectoryTreeView
PathType	Desktop
NotifyComponent	DCBX_2

16. Select the right hand *Explorer* component (DCBX_2) and set its properties:

Property	Value
DisplayStyle	GeneralListView
FileIncludeMask	
	*.bmp,*.jpg,*.jpeg,*.gif,*.tif,*.tiff

The Explorer DBCX_2 will display the defined file types only. 17. Save your changes.

#### Step 3. Make the Find Image Form Useful

1. Switch to the *Source* tab. Define an *uImageSelected* event with a map for input of class STD_STRNG:

Define_Evt Name(uImageSelected) Define_Map For(*input) Class(#std_strng) Name(#uFilename)

The event, when signalled, will pass the full filename for the selected image.

2. Create an *ItemGotSelection* event for Explorer DCBX_2, using the *Design* view and the context menu on DCBX_2:

Evtroutine Handling(#DCBX_2.ItemGotSelection) Options(*NOCLEARMES Endroutine

3. This routine should set the *FileName* property for the Image component (IMGE_1) from the Explorer DCBX_2, Path and Filename Your code should look like the following:

#IMGE_1.fileName := #DCBX_2.path + #DCBX_2.filename

- 4. In the *PHBN_OK.Click* event routine
  - a. Signal the *uImageSelected* event.
  - b. Populate the *uFilename* parameter using the DCBX_2 *Explorer* properties.
  - c. Then close the form.

Your *OK* push button *Click* event should look like the following:

Evtroutine Handling(#PHBN_OK.Click) Signal Event(uImageSelected) Ufilename(#DCBX_2.path + #DCBX_2.filenan #com_owner.closeForm Endroutine

5. Complete the *PHBN_CANCL* Push button Click event, to close the form:

Evtroutine Handling(#PHBN_CANCL.Click) #com_owner.closeForm Endroutine 6. Compile your form.

#### Step 4. Create the Employee Image Command Handler

1. Create a new *Reusable Part / Panel*:

#### Name: iiiVFW16

#### Description: Employee Image CH

Make the Panel size approximately *Height* **350** and *Width* **500**.

- 2. Give iiiVFW16 an ancestor of VF_AC010.
- 3. In the *Design* view, from the *Designer* ribbon, give the form an *Attachment manager*:
- 4. Drop a Panel onto the right hand side. And change its *Name* to **BUTTON_PANEL**. Adjust its *Width* to allow buttons to be added.
- 5. Drop a Panel onto the center of the form and change its *Name* to **IMAGE_PANEL**.
- 6. Select the BUTTON_PANEL and add a *Flow Down* manager.
- 7. Drop two push buttons onto the BUTTON_PANEL.
- 8. With the BUTTON_PANEL selected, use the *Layout Helper* tab to set *Category / Margins*. Use the **All** settings to position the buttons approximately in the middle of the panel horizontally. The setting required will depend on the *Width* of your BUTTON_PANEL. **12** pixels should be about correct.

												1		
								_						
1								۰.						
		•	n										1	
		•	11										ŀ.	•
			12							-			J.,	
1			-		-							-		
		•	1	•	•	•	•	•	•	•	•	•	÷.,	
•		•	11										ŀ.	
			11											
			-	-	-	-	-	-	-	-	-	-	÷.,	
		•	•	•	•	•	•	•	•	•	•	•	•	
		•	•	•	•	•	•	•	•	•	•	•	•	
			•	•	•	•	•	•	•	•	•	•	•	
-1	-	٨.	n.,											

9. Set up two *Push Button* properties:

#### Property Value for the first push button Value for the second push button

Name	PHBN_SAVE	Find
Caption	Save	PHBN_FIND

- a. Create a Click event for the *Save* button.
- b. Create a Click event for the *Find* button.
- 10. Save your reusable part.
- 11. Drop a *Group Box* into the center of the IMAGE_PANEL. Resize it and change its *Caption* to **Employee Image**.
- 12. Drop an *Image* component into the center of the *Group Box* and resize it to fill the Group Box.

Your design should now look like the following:



13. Save your reusable part.

## **Step 5. Complete the Image Command Handler**

1. In the *Design* view, drag and drop your Find Employee Image form (iiiVFW15) onto the Employee Image CH reusable part (iiiVFW16).

This will create a *Define_Com* for iiiVFRW15. Select the *Source* tab and change the *Name* of iiiVFW15 to **Find_Image**:

Define_Com Class(#IIIVFW15) Name(#Find_Image) Componentversion(2)

2. An Instance List command handler is invoked by the Framework, when an entry is selected in the business object Instance list and its *uExecute* method is invoked.

Create the *uExecute* method routine. This must redefine the method which is already defined in the ancestor VF_AC010. It should invoke the *uExecute* method in the ancestor.

Your code should look like the following:

```
Mthroutine Name(uExecute) Options(*redefine)
#com_ancestor.uExecute
Endroutine
```

- 3. Your *uExecute* logic, needs to do the following:
  - a. Get the employee number for the current instance list entry
  - b. Fetch the employee image field (iiiEmpImg) from file iiiEmpImages
  - c. Set the image component filename property from the retrieved employee image
  - d. Handle a not found error, when the employee has no image.

Your new code should look like the following:

```
#avlistmanager.getCurrentInstance Akey1(#empno)
Fetch Fields(#iiiempimg) From_File(iiiEmpImages) With_Key(#empno) Val_I
If_Status Is(*okay)
#IMGE_1.fileName := #iiiempimg.fileName
Else
#IMGE_1.fileName := *null
#IMGE_1.updateDisplay
Endif
```

4. The *Find* button *Click* event, simply needs to invoke the *Find Employee* image form, as a modal form.

Your completed Find button *Click* event should look like the following:

Evtroutine Handling(#PHBN_FIND.Click) #Find_Image.ShowModalForm Endroutine

5. When an image is selected in the *Find Employee Image* form, the user will click the *OK* button. This will signal the *uImageSelected* event, passing the image path and filename.

In the *Employee Image* command handler, add an event handling routine, which sets the image filename (IMGE_1.filename).

Your code should look like the following:

Evtroutine Handling(#Find_Image.uImageSelected) Ufilename(#FileName) #IMGE_1.fileName := #FileName Endroutine

- 6. The *Save* button *Click* routine, needs to do the following:
  - a. Set the field value for iiiEmpImg to the image filename property
  - b. check for an existing entry in file iiiEmpImages
  - c. Update or Insert to the file iiiEmpImages as appropriate.

Your code should look like the following:

Evtroutine Handling(#PHBN_SAVE.Click) #iiiempimg := #IMGE_1.fileName Check_For In_File(iiiEmpImages) With_Key(#empno) Val_Error(*next) If_Status Is(*equalkey) Update Fields(#iiiempimg) In_File(iiiEmpImages) With_Key(#empno) Else Insert Fields(#empno #iiiempimg) To_File(iiiEmpImages) Endif Endroutine

7. Compile your command handler iiiVFW16.

# **Step 6. Plug In and Test the Image Command Handler**

- 1. Open your Framework.
  - a. Open the *Employees business object* properties
  - b. Select the Commands Enabled tab
  - c. Select the *Images* command handler and plug in the reusable part, iiiVFW16.
  - d. Save and restart your Framework.

You could use the *Find Employee Images* form to find your own images of any supported type. A suitable set of images is available for these exercises on the *LANSA / Support / Documentation* web site (see http://www.lansa.com/support/docs/index.htm). Look under the Visual LANSA group for VFW_WAW Workshop – Extra Files. The zip file contains a folder \PHOTOS which contains 16 small facial

The zip file contains a folder \PHOTOS which contains 16 small facial images (80 x 90 pixels).

- 2. Select the *Images* tab for an employee. Use the *Find Employee Image* form to locate C:\LANSA\PHOTOS folder. Select an image and click *OK*, which should close the *Find Image* form.
- 3. Click the *Save* button on the *Images* command handler.
- 4. Save images for a number of employees and check that images can then be retrieved and displayed when an employee is selected in the Instance list.

### Summary

#### **Important Observations**

• Two Explorer components can be linked so that output from one can be shown in the other.

### **Tips & Techniques**

• Your component can retrieve the windows path and file name from the Explorer component.

# What You Should Know

- How to implement the Explorer component.
- What are the different views it can provide.
- How to store images in the database as a BLOB field.

# VFW082 – Toolbars, Menus and Pop–up Menus

The VL Framework can provide most of the toolbar, menu and pop-up menu facilities your application will require. Setting properties in VLF for business objects and command handlers enables you to configure the menus, toolbars and associated icons.

# Objectives

The objective of this exercise is for you to build a standalone form which is not part of a Framework application. This demonstrates how to implement menus, toolbars and pop-up menus in your own code.

					1	Me	enu	is ar	nd T	oolb	ars	5				-				×
File	Edit	View	Help																	
~	Fill List																			
	Clear Li	st																		
	Exit			artme	ent	De	scrip	otion					0.000							0.00
-	100		-	inistr	atio	n														
VA	UD		Inte	mal A	ludi	iting	9													
V F	LT		Flee	et Adr	nini	stra	tion													
- G	iAC		Gro	up Ac	000	unt	s													
~ IN	NF		Info	rmatio	on S	Ser	vices	s												
VL	EG		Leg	al																
V M	4IS		Mg	nt. Inf	form	nati	on													
V M	4KT		Ma	keting	g															
VS	D		Sal	es & D	Distr	ribu	tion													
🗸 T	RVL		Tra	vel																
												0	7/0	6/20	13	v 1	6:0	06:3	8	1

To achieve this objective you will complete the following:

Step 1. Create Menu and Toolbars Form.

Step 2. Make the Menus Useful

Step 3. Add a Pop-up Menu to the Status Bar

Summary

# **Before You Begin**

This exercise is standalone and doesn't depend on earlier exercises.

### Step 1. Create Menu and Toolbars Form.

1. Create a new *Form / Basic Form*:

Name: iiiVFW17

Description: Menus and Toolbars

2. On the *Controls* tab, select *All Controls* and drag and drop a *Menu Bar* component to the top of the form. The *Menu Bar* component will always attach to the top of the form.

Note that the first Menu Item is displayed at the top left and can be edited in the *Design* view.

	3																																														-			]		>	<	
Ite	en	n1	1		ſ																																																	
-																																																						
-				• •				-				-						-		-		-								• •		-																		-				
										• •							τ.												1				6. 6							6.1						1.1					1.1		1.1	1.1
-																		-				-																																1.1
-		-		• •	•			-				-						-	• •			- 1							•			-						• •	-											-			1.1	
							1.1				. 1						τ.	× )							1				1.1																	1.11			1.1		1.1		1.1	1.1
-								-				-										-																	-					-	- 1					-				
-				• ;	-	120	~	2	1		-	-	~	N	1.2		£	-	-	-	20	-	43	4	5	1	1.	10	1		1	-		3	~	1	1	-	-	2	~	1	-	1		2	200	-	-		1000	1	-	-3

3. Review the source code. Note that two components have been defined:

A Menu Bar (MBAR_1)

A Menu Item (MITM_1) which has a parent of MBAR_1

Define_Com Class(#PRIM_MBAR) Name(#MBAR_1) Parent(#COM_OWNF Define_Com Class(#PRIM_MITM) Name(#MITM_1) Caption('Item1') Displa

4. Return to the *Design* view and define the menu. In the following steps you will define the menu as shown in the following table:

Menu	Option
File	Fill List
	Clear List
	-
	Exit
Edit	Сору
	Delete

View	Icon
	List
	Report
	Small Icon
Help	Contents

- 5. Follow these steps:
  - a. Replace Item1, with &File. The "F" will be the menu shortcut key. At run time, *Alt* + *F* will open the File menu.
  - b. Press *Enter* to create the next menu item. Type F&ill List. Typing "i" with the File menu open will select the Fill List option.
  - c. Press *Enter* and the editor will move the input menu items to top of the *Design* tab. Continue to define the menu here.
  - d. Type C&lear List and press *Enter*.
  - e. Type a dash character (-) into the new item. This makes the menu item a divider. Press *Enter*.
  - f. In the new menu item, type E&xit and save your form.

You have created MITM_1 and its sub-menu items.

6. Once again review your source code:

Define_Com Class(#PRIM_MBAR) Name(#MBAR_1) Parent(#COM_OWNF Define_Com Class(#PRIM_MITM) Name(#MITM_1) Caption('&File') Displa Define_Com Class(#PRIM_SMNU) Name(#SMNU_1) Parent(#MITM_1) Define_Com Class(#PRIM_MITM) Name(#MITM_2) Caption('F&ill List') Di Define_Com Class(#PRIM_MITM) Name(#MITM_3) Caption('C&lear List') ] Define_Com Class(#PRIM_MITM) Name(#MITM_4) Caption('E&xit') Displa Define_Com Class(#PRIM_MITM) Name(#MITM_5) Caption('-') Displayposition(3) Parent(#SMNU_1)

You now have a sub menu item (SMNU_1) with four menu items belonging to the sub menu which have a parent of SMNU_1.

7. Compile the form and execute it. Check that your menu looks as expected.

File		
	Fill List	
	Clear List	
	Exit	
-		

- 8. Test the shortcut keys.
- 9. Close the form.
- 10. Switch to the *Design* view. Position the cursor in the File menu option and press the *Tab* key. A new menu element will be displayed below:

I	B	]																																					L	-	-			-			2	x		J
Γ	F	ile	2																																															
F	:	:	:	:	:	:	:			:	:		:	:	:	:	:			:	:	:	:	:											:	:	:	:	:	:	:	:	:	:	:			:	:	:
ŧ			· · ·					 	5	 			• • • •	· · ·				 5	5						 ;	 ~	v	 	 ~	~	~	-	~		 							:				~		-	• • •	÷

11. Type **&Edit** in the new element and press *Tab*. Type **&**View and press *Tab*, and type **&Help**.

Your menu should look like the following:



- 12. Save your form.
- 13. Click in the Edit menu item. Ensure the cursor is positioned at the right hand side and press *Enter*. A new submenu will be displayed below.

<u>F</u> ile	&Edit View Help
Lin	······································

- 14. Type **&Copy** and press *Enter* and type **&Delete** in the new menu item.
- 15. Position your cursor cursor in the *View* menu item. Move the cursor to the right and press *Enter* to create a new sub menu. You will now define four sub-menu items for the View menu:
  - a. In a new menu item, type **&Icon** and press *Enter*.
  - b. In a new menu item type **&List** and press *Enter*.
  - c. In a new menu item type **&Report** and press *Enter*.
  - d. In a new menu item type **&Small Icons**.
- 16. Position your cursor in the Help menu item and move the cursor to the right. Press *Enter*. In the new menu item type **&Contents**.
- 17. Compile your form and execute it.



- 18. Test all menu options and shortcuts.
- 19. Close your form.

# **Step 2. Make the Menus Useful**

- 1. Add a *Status bar* to the form.
- 2. Add a *List view* component, resize it as shown. Leave space below the menu bar, where you will be adding a toolbar.



- 3. On the *Repository* tab, find the file DEPTAB and add columns to the list view for DEPTMENT and DEPTDESC. Resize the columns to use all the space available. Note that the DEPTDESC column can be given a *WidthType* property of **Remainder**.
- 4. In the *Design* view, position the cursor in the File menu item and press *Enter* to display the sub menu.
- 5. On the *Details* tab, select the *Events* tab and create a *Click* event routine.
- 6. Create *Click* events for the *Clear List* and *Exit* menu items.
- 7. Switch to the *Source* tab and complete the *Click* event for the **Fill List** menu item based on the following:
  - Clear the List view.
  - Populate the List View with all records from the file DEPTAB.
  - Add an image component to each list item.
  - Set menu item 2 checked to true, if not checked.

• Set menu item 3 to checked false

#### Note:

- List items (*CurrentItem*) have an *Image* property. This may have an Image component associated with them. The image could be set dynamically dependent on the data in each row.
- An Icon component VI_CHECK which already exists in the Repository should be used for this exercise.
- Feature Help (*F2*) is available for any component for you to investigate its Properties, Events and Methods in detail.

Your code should look like the following:

```
* Fill List
Evtroutine Handling(#MITM_2.Click)
Clr_List Named(#LTVW_1)
Select Fields(#LTVW_1) From_File(deptab)
Add_Entry To_List(#LTVW_1)
* Set a reference from VI_CHECK image component to list item image proper
#LTVW_1.CURRENTITEM.IMAGE <= #VI_CHECK
Endselect
#MITM_2.checked := *Not #MITM_2.checked
#MITM_3.checked := false
Endroutine
```

8. Add the basic code for the *Click* event for the *Clear List* and *Exit* menu items.

Your code should look like the following:

```
* Clear List
Evtroutine Handling(#MITM_3.Click)
Clr_List Named(#LTVW_1)
Endroutine
* Exit
Evtroutine Handling(#MITM_5.Click)
#com_owner.closeForm
Endroutine
```

9. The *Clear List* and *Fill List Checked* property should be handled similarly to step 7.

Add logic to check *Clear List* (MITM_3) and uncheck *Fill List* (MITM_2) to the *MITM_3.Click* event. Changes are highlighted in red.

Evtroutine Handling(#MITM_3.Click) Clr_List Named(#LTVW_1) #MITM_3.checked := *Not #MITM_3.checked #MITM_2.checked := false Endroutine

10. Compile and test your form. Your list should look like the following:

	Menus and Toolbars – 🗖 🗙
File Edit View	Help
✓ Fill List	
Clear List	
	artment Description
Exit	inistration
🗸 AUD	Internal Auditing
🗸 FLT	Fleet Administration
🗸 GAC	Group Accounts
🗸 INF	Information Services
🗸 LEG	Legal
🗸 MIS	Mgmt. Information
🗸 МКТ	Marketing
🗸 SD	Sales & Distribution
V TRVL	Travel

11. In the next few steps you will add logic to handle the **View** menu items. The *List View* component has a *ViewStyle* property. Select the *List View* and on the *Details* tab review the list of values available for the *ViewStyle* property:

<u> </u>	-
🖻 TabStop	True
🖻 Тор	46
ViewPopupMenu	*NULL
( ViewStyle )	Report 💌
Visible	Icon
VisualStyle	List
VisualStyleOfParent	SmallIcon
🖻 Width	463 👻

The **View** menu items will control the *List View ViewStyle* property.

12. This time, use the *Outline* tab to access and select the menu items. This tab is usually open, but if necessary select it from the *Home* ribbon, *Views* menu.



On the *Outline* tab, expand sub-menu items and note that the appropriate menu is opened in the *Design* view.

13. Select the **View** menu items and create a *Click* event for each item. You can use the context menu on the *Outline* tab to create the *Click* events.

<u>File</u> <u>E</u> dit	View	Help	
	84	con	
	Lis	t	
Desertes	<u>R</u> e	port	
	Sm	nall Icons	
	r	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~~

14. Add logic to each new click event routine to set the List View.ViewStyle property as required. Your finished code should look like the following:

```
* Icon
Evtroutine Handling(#MITM_11.Click)
#LTVW_1.viewstyle := icon
Endroutine
* List
```

Evtroutine Handling(#MITM_12.Click) #LTVW_1.viewstyle := list Endroutine * Report Evtroutine Handling(#MITM_13.Click) #LTVW_1.viewstyle := report Endroutine * Small Icons Evtroutine Handling(#MITM_14.Click) #LTVW_1.viewstyle := smallIcon Endroutine

- 15. Compile and test your form. Use the *View* menu options to change the appearance of the list.
- 16. In the next few steps you will add a Toolbar to the top of your form and implement a number of toolbar buttons.
  - a. Drag and drop a *Group Box* component to the top of the form below the *Menu Bar* and resize it:
  - b. Give the *Group Box* a *ThemeDrawStyle* property of **Toolbar**.
  - c. Set the *Group Box Height* property to **48** pixels.



**Note:** Remember you can move components accurately using the *Ctrl+Cursor* keys.

- 17. Use the *Design* ribbon to add a *FlowAcross* manager to the *Group Box*.
- 18. Use the *Layout Helper / Layout Manager Details* tab to set the *Margins Category*. Use *All* to set the value to **3** pixels.
- 19. Add four *Toolbar* buttons to the *Group Box*. Use the *Shift* + *Left Mouse* to select all the toolbar buttons and set their *ButtonStyle* property to **FlatButton**.


When you focus elsewhere the toolbar buttons will not be visible. You can still click on their position to select one. Select the third button and change its *ButtonStyle* to **Separator**.

20. Select each Toolbar button and set the *Image* and *Hint* properties as follows:

#### Button Image Hint

SPBN_1 xImageNew16 Fill the list view

SPBN_2 xImageCut16 Clear the list view

SPBN_4 xImageExit16 Close the form

21. Since the toolbar duplicates some of the menu options, you should add the new toolbar button click events to the existing logic. Your completed code should look like the following. Changes are highlighted in red.

```
Evtroutine Handling(#MITM_2.Click #SPBN_1.click)
Clr_List Named(#LTVW_1)
Select Fields(#LTVW_1) From_File(deptab)
Add_Entry To_List(#LTVW_1)
#LTVW_1.CURRENTITEM.IMAGE <= #VI_CHECK
Endselect
#MITM_2.checked := *Not #MITM_2.checked
#MITM_3.checked := false
Endroutine
Evtroutine Handling(#MITM_3.Click #SPBN_2.click)
Clr_List Named(#LTVW_1)
#MITM_3.checked := *Not #MITM_3.checked
#MITM_2.checked := false
Endroutine
```

Evtroutine Handling(#MITM_5.Click **#SPBN_4.click**) #com_owner.closeForm Endroutine

22. Compile your form.



23. Test your form, especially the menu options and toolbar buttons.

# Step 3. Add a Pop-up Menu to the Status Bar

A *Pop-up Menu* component can be associated with most of the visual components on a form. A right click on this visual component will then display the "context menu".

You will add date and time fields to the right side of the *Status* bar and the *Pop-up Menu* will then control whether the date and time fields are visible.

- 1. From the *Repository* tab, drag and drop the STD_DATEX field into the right hand side of the *Status Bar*.
- 2. Hide its label by changing its *MarginLeft* property to **0** and then reduce its *Width* as required..
- 3. Add field STD_TIMX to the right hand side of the *Status Bar*. Hide its label and adjust its width. Your form should now look like the following:



**Hint:** If you have difficulty adding the Time field, drop it onto the *Status Bar up* / *down* buttons and it will be positioned on the right hand side of the status bar.

- 4. Set both the date and time fields *ReadOnly* property to **true**.
- 5. In this step you will ensure both the date and time fields have a value.
  - a. Add logic to the form's Initialize event routine to set STD_DATEX to current date:

#STD_DATEX := *date

b. Drag and drop a *Timer* component onto your form. This is a non-visual component. The component definition is:

Define_Com Class(#PRIM_TIMR) Name(#TIMR_1)

c. In the source editor, select the timer component name and use the context menu to create a click event routine.

Endroutine * Fill List Evtroutine Handling(#MITM_2.Click #SE Clr_List Named(#LTVW_1) Select Fields(#LTVW_1) From_File(de Add_Entry To_List(#LTVW_1) #LTVW_1.CURRENTITEM.IMAGE <= #VI Endselect #MITM_2.checked := *Not #MITM_2.che #MITM_3.checked := false Endroutine * Clear List	Delete Component Copy Component Cut Component Cut Ctrl+X Copy Ctrl+C Paste Ctrl+V Command Assistant F4 Set As Breakpoint F9 Expand All Collapse All
Job Status Description	

d. Add logic to the timer click event to set STD_TIMX to current time: Evtroutine Handling(#TIMR_1.Tick) Options(*NOCLEARMESSAGES *NO( #STD_TIMX := *time

Endroutine

**Note:** In this case, there is no need for other code. The standard timer behavior is to fire a click event every second. This is defined by its *Interval* property of **1,000**. Setting the *Interval* to **0** will stop the timer.

The Timer also has a Start and Stop method.

- 6. Compile and execute your form to check that you are displaying a date and time.
- 7. Close your form.
- 8. Continue on the *Design* tab, by dragging and dropping a *Pop-up Menu* component onto the *Status Bar*.

The *Pop-up Menu* will be shown at the top of the *Design* tab:

esign Source Rep	ository Details	Repository Help Cri	oss References		
Item1				1	
File Edit View	Help	Pop-up menu Bai	ı for Status r		×
Department Code	Department De	scription			Щ
ABCD	aAbBcCdDeEfF	gGhHiljJ			

- 9. Define two menu items, **Show &Date** and **Show &Time**. Create a *Click* event for the new menu items.
- 10. In the *Design* view, select the *Status Bar* and review its properties. Note that it now has a *PopUpMenu* property of **PMNU_1**.

Details			Design Source Repository Details
STBR_1		- 🖬	E
Properties Events Meth	ods		File Edit View Help
E • ?			
P Height	24	^	🖌 🤞 🕺
🖻 Hint			Department Code Department D
HintPopup	*NULL		ABCD aAbBcCdDeEf
HintShow	True		
HintShowOfParent	True		
HintTitle			
LayoutStyle	Attach		
🖻 Left	0		
MessagePosition	1		
MouseOverStyle	*NULL		
Name	STBR_1		
Opacity	100		
Owner	#IIIVFW17		
Parent	#IIIVFW17		
Popup	*NULL		
PopupMenu	#PMNU_1		aAbBcCdDeEfFgGhHiljJkKILmMnNo
PrivateStyle-	MULL		

Date and time are initially visible.

11. On the *Design* tab, select the *Status bar* and use the context menu to *Edit* 

Pop-Up Menu.

- a. Set both *Pop-up Menu* items *Checked* property to **True**.
- b. Complete the *Click* event code for the *Status Bar Pop-up Menu* component, to switch the date and time from *Visible* = true / false and *Menu Item Checked* = true / false.

Your code should look like the following:

Evtroutine Handling(#MITM_16.Click) #STD_DATEX.visible := *Not #STD_DATEX.visible #MITM_16.checked := *Not #MITM_16.checked Endroutine Evtroutine Handling(#MITM_17.Click) #STD_TIMX.visible := *Not #STD_TIMX.visible #MITM_17.checked := *Not #MITM_17.checked Endroutine

12. Compile and test your form.

# Summary

# **Important Observations**

- If you are developing applications based on VL Frameworks, then most of your menus and toolbars can be provided by the Framework.
- Check the *Repository* under *Resources / Bitmaps* for a range of suitable images. These are mainly PNG format images in various sizes (for example, 16x16, 32x32, 64x64 etc.).

# **Tips & Techniques**

- In this exercise all the menu and spin button items were left with default names. E.g. MITM_1. In a real application it is good design and will simplify maintenance if you give menu items meaningful names.
- The templates VL_BBMNUBR and VL_FBBMNUB can be used to generate a menu bar in a form that you can easily adapt to suit your own requirements. You should generate from the template into an RDML-enabled form and then change the form to be RDMLX-enabled.

# What You Should Know

• How to implement menus, toolbars and pop-up menus.

# VFW084 – A Business Object Browser and Detailer

# Objectives

In this exercise you will create two reusable parts which work together.

- The Business Object Browser displays a list skills for an employee selected in the instance list.
- The Business Object Detailer displays details for an employee's skill record, selected in the Business Object Browser.

If developing the application using VL Frameworks, there are other ways this functionality could be achieved. For example the employees instance list could contain another level for employee skills. See the shipped *HR Demo Application / Organisations* for an example which illustrates this concept.

The objective for this exercise is to show how to design and build a simple application with two components which interact with each other.

🍰 Employ	ee : Sl	kills 2 (A1007-SNELL (	GEORGE)				x
Details Skill Code ADMIN1 ADMIN2 CS INTRO KEY MANA MANA MARKET1 MARKET3 REL	P P P P P D F D D D D	rief Notes 1 Images Comment Met requirement Met requirement Met requirement Met requirement Met requirement	Notes         Skills 2           Date Acquired           25/03/98           1/05/98           5/05/98           5/02/98           5/02/98           15/03/98           2/05/98           2/05/98           3/05/98           4/05/98	Skill	Is Skill Code Grade Obtained for Skill Comment on skills acquired Date Skill Acquired Save Dele	ADMIN1 D Met requirement 25/03/98	

To achieve these objectives you will complete the following steps:

Step 1. Create Employee Skills Command Handler

Step 2. Create Business Object Detailer.

Step 3. Complete the Skills Browser

Summary

# **Before You Begin**

You should complete VFW030, VFW040 and VFW042 before starting this exercise.

# **Step 1. Create Employee Skills Command Handler**

This is the **Business Object Browser**, which will be referred to as the **BOB**.

1. Create a new *Reusable Part / Panel*:

### Name: iiiVFW18

### Description: Employee Skills Browser.

Give the reusable part an ancestor of VF_AC010.

- 2. From the *Design* tab, give the reusable part a *Split Vertically* manager. On the *Layout Helper* tab give this a *Divider Style* of **Raised**. The *Vertical Splitter* defines two panels *PANEL_1* and *PANEL_2*.
- 3. Select the left side Panel and change its *Name* to **LEFT_PANEL**. Change the right side Panel's *Name* to **RIGHT_PANEL**.
- 4. Select LEFT_PANEL and give it an *Attachment layout* manager.
- 5. Drop a *List View* into the center of LEFT_PANEL. Change the List View's *Name* to **SKILL_LIST**.
- 6. On the *Repository* tab, select the file PSLSKL and drag and drop SKILCODE, GRADE, COMMENT and DATEACQ into the list view SKILL_LIST
- 7. Select each column heading and change the *Caption* and *Caption Type* as shown:

Field	Property	Value
SKILCODE	Caption	Code
	CaptionType	Caption
GRADE	Caption	Grade
	CaptionType	Caption
COMMENT	Caption	Comment
	CaptionType	Caption
DATEACQ	Caption	Date Acquired
	CaptionType	Caption

8. Create a *uExecute* method routine, which redefines the ancestor method. Add logic to execute the ancestor *uExecute* method:

Mthroutine Name(uExecute) Options(*redefine) #com_ancestor.uExecute Endroutine

9. Add logic to get the current instance list entry and return AKey1.

Mthroutine Name(uExecute) Options(*redefine) #com_ancestor.uExecute **#avlistmanager.getCurrentInstance Akey1(#empno)** Endroutine

10. The SKILL_LIST list view will need to be built, initially when the command handler is invoked and also whenever the Business Object Detailer has added or changed an employee skill. Create a subroutine to populate the list view. Your code should look like the following:

# Subroutine Name(buildlist)Clr_List Named(#SKIIL_LIST)Select Fields(#SK Endroutine

11. Execute the subroutine from the *uExecute* method:

Mthroutine Name(uExecute) Options(*redefine) #com_ancestor.uExecute #avlistmanager.getCurrentInstance Akey1(#empno) Execute Subroutine(buildlist) Endroutine

- 12. Compile your new command handler.
- 13. Open your Framework and plug in iiiVFW18 as the command handler for *Employee* business object, command *Skills 2*.
- 14. *Save and Restart* your Framework. Select an employee and then select the Skills 2 command handler tab. Check that the list view is populated as expected. Not all employees have skill records.

Employee : S	kills2 (A1015-Woods,	Bradley		
Details 📎	Brief Notes 🥑 Image	Notes 😕 Skills 🦻 Skills 2		
Code	Grade	Comment	Date Acquired	
ADMIN 1	D	Met requirement	25/03/98	
ADMIN2	F		2/05/98	
CS	P		5/05/98	
ECD	P		1/05/98	
INTRO	P	Met requirement	5/02/98	
KEY	P	Met requirement	5/02/98	
MANAGE1	D	Met requirement	15/03/98	
MARKET1	D	Met requirement	25/03/98	
OTH	D		4/05/98	
SHORT	P		3/05/98	

15. Close the Framework.

# Step 2. Create Business Object Detailer.

In this step you will create a reusable part which will be displayed on the RIGHT_PANEL in iiiVFW18.

This is the **Business Object Detailer** for an employee skill, which will be referred to as the **BOD**.

The BOB will invoke the BOD to display details or create a new employee skill. The BOD will signal *uSkillChanged* when an employee skill has been updated, created or deleted.

When an item in the SKILL_LIST is selected, the BOB will invoke the BOD's *uShow* method, passing employee number and skill code.

When displaying an employee skill the BOD will enable a *Delete* button.

The BOB will have a *Pop-Up Menu* on the employee skills list, which will invoke the BOD's *uNew* method passing the employee number..

1. Create a new Reusable Part / Panel:

### Name: iiiVFW19

### Description: Employee Skill Detailer

**Note:** This component does not interact directly with Framework components. It therefore does not need to have VF_AC010 as its ancestor.

2. In the *Design* view resize the panel to approximately *Height* = **350** and *Width* = **370**.

Use the *Design* ribbon to give iiiVFW19 an *Attachment* layout manager.

- 3. Drop a Panel at the bottom of the main panel. Change its *Name* to **BUTTON_PANEL**.
- 4. Select BUTTON_PANEL and give it a *Flow Across* layout manager.
- With the BUTTON_PANEL selected, use the *Layout Helper / Layout Manager Details* tab and select *Category* = Margins and set *Left* and *Top* = 10 pixels.
- 6. Drag and drop two *Push Buttons* onto BUTTON_PANEL. Set up their properties as:

# **Caption** Name

Save PHBN_SAVE

#### Delete PHBN_DELETE

- 7. Create a *Click* event routine for each button.
- 8. Drop a *Panel* into the center of the main Panel and change its *Name* to **DETAIL_PANEL**.
- 9. Give DETAIL_PANEL a *Flow Down* layout manager. On the *Layout Helper* / *Layout Manager Details* tab, select *Category* = **Margins** and use the *All* setting to set margins of **7** pixels.
- 10. On the *Repository* tab, find the file PSLSKL and drop fields SKILCODE, GRADE, COMMENT and DATEACQ onto DETAIL_PANEL.
- 11. Save your component.
- 12. Create a *uShow* method routine, with input parameters based on EMPNO and SKILCODE.

Mthroutine Name(uShow)

Define_Map For(*input) Class(#empno) Name(#i_empno) Define_Map For(*input) Class(#skilcode) Name(#i_skill) Endroutine

13. Define a character work field, REQUEST, *Length* **3**.

Define Field(#request) Type(*char) Length(3)

- 14. Complete the *uShow* routine, which should:
  - a. Set Request to DET
  - b. Set EMPNO and SKILCODE from the values input to this method
  - c. Fetch all employee skills from file PSLSKL.

Your code should look like the following. Changes are highlighted in red.

Mthroutine Name(uShow) Define_Map For(*input) Class(#empno) Name(#i_empno) Define_Map For(*input) Class(#skilcode) Name(#i_skill) #request := DET #empno := #i_empno #SKILCODE := #i_skill Fetch Fields(*all) From_File(pslskl) With_Key(#empno #SKILCODE) Val

#### If_Status Is_Not(*OKAY) Message Msgtxt('Employee skill not found') Endif Endroutine

15. Create a *uNew* method routine with an input parameter i_empno, based on EMPNO.

Mthroutine Name(uNew) Define_Map For(*input) Class(#empno) Name(#i_empno) Endroutine

- 16. Define a Group_by, named SKILDATA for fields SKILCODE, GRADE, COMMENT and DATEACQ, at component level.
- 17. Complete the *uNew* method routine which should:
  - a. Set request to NEW
  - b. Set EMPNO to the value input to this method
  - c. Set SKILDATA to *null

```
Mthroutine Name(uNew)
Define_Map For(*input) Class(#empno) Name(#i_empno)
#empno := #i_empno
#request := NEW
#skildata := *default
Endroutine
```

18. Define an event *uSkillChanged*.

define_evt NAME(uSkillChanged)

- 19. Complete the *Save* push button *Click* event routine, which should:
  - a. Handle a request of DET or NEW using a CASE/ENDCASE.
  - b. When request is DET, UPDATE all fields in PSLSKL
  - c. Check status code and signal *uSkillChanged*
  - d. When request is NEW, INSERT all fields to PSLSKL
  - e. Check status code and signal *uSkillChanged*.

Your code should look like the following:

```
Evtroutine Handling(#PHBN_SAVE.Click)
Case (#request)
When (= DET)
Update Fields(*all) In_File(pslskl) Val_Error(*next)
If_Status Is(*okay)
Signal Event(uSkillChanged)
Else
Message Msgtxt('Employee skill update failed')
Endif
When (= NEW)
Insert Fields(*all) To_File(pslskl) Val_Error(*next)
If_Status Is(*okay)
Signal Event(uSkillChanged)
Else
Message Msgtxt('Add Employee skill failed')
Endif
Endcase
Endroutine
```

20. Complete the *Delete* push button *Click* event routine, which should:

- a. Delete from the file PSKSKL
- b. Check status and signal *uSkillChanged* and issue an error message:

Evtroutine Handling(#PHBN_DELETE.Click) Delete From_File(pslskl) Val_Error(*next) If_Status Is(*okay) Signal Event(uSkillChanged) Else Message Msgtxt('Employee skill deletion failed') Endif Endroutine

21. Compile reusable part iiiVFW19.

# Step 3. Complete the Skills Browser

1. Switch to the reusable part iiiVFW18 (Employee Skills Browser) in the editor. In the Design view, select the panel RIGHT_PANEL. Use the *Layout Helper* tab and give this panel an *Attachment* layout manager by selecting the existing manager ATLM_1 in the drop down:

Layout Help	er				
Layout Mana	ged (	Comp	pone	nt	Ē
RIGHT_PANE	L				-
Layout					<b>Ľ</b> ×
ATLM_1					-
Children	0	×	U	×	Layout Manager Details Child Details A
-				~	Category Processing Order

- 2. Drop the reusable part iiiVFW19 into the center of the Panel RIGHT_PANEL. Use the *Details* tab to change its *Name* to **SKILL_DETLS**.
- 3. Select the component SKILL_DETLS on the RIGHT_PANEL and use *Details / Events* to create a *uSkillChanged* event routine. Add logic to execute the *BuildList* subroutine.
- 4. Select the SKILL_LIST list view on the LEFT_PANEL and create an *ItemGotSelection* event routine.

Add logic to invoke the *uShow* method in SKILL_DETLS, passing employee number and skill code. Your code should look like the following:

Evtroutine Handling(#SKILL_LIST.ItemGotSelection) Options(*NOCLEARM Invoke Method(#SKILL_DETLS.ushow) I_Empno(#empno) I_Skill(#skilcode Endroutine

5. On the *Design* tab, select the SKILL_LIST and use the context menu to add a *Pop-Up Menu* component.

Enter *New* in the menu item displayed at the top of the *Design* tab and create a *Click* event for it.

6. Complete the menu item *Click* event by adding code to invoke the *uNew* method in SKILL_DETLS passing employee number.

Evtroutine Handling(#MITM_1.Click) Invoke Method(#SKILL_DETLS.uNew) I_Empno(#empno) Endroutine

- 7. Compile the reusable iiiVFW18.
- 8. Execute your Framework, and ensure you can use the *Skills 2* command handler for an employee to update, delete and create a new skill.

At the moment this command handler has a very basic interface. In the next exercise you will make a number of improvements to it.

### Summary

# **Important Observations**

- Your components will contain your own application, properties, events and methods.
- Properties, events and methods are "published" unless they are defined as *Access(*private)*.
- In this simple component model, the "browser" displays all skill entries (for an employee) and the "detailer" enables the user to change or delete a skill.

# Tips & Techniques

- The business object detailer signals an event telling the business object browser an employee skill has changed.
- The business object browser passes employee number and skill code into the *uShow* method in the business object detailer.

# What You Should Know

• How to design and implement a simple multi-component application.

# VFW090 – Field Visualizations

- The *Repository* defines fields as components. An important part of this component definition is the field visualization.
- Most fields are visualized by default as a simple edit box.
- Other field visualizations may be added, such as a radio button set, a dropdown together with a static picklist, check box, multi-line edit box and date prompt.

The following example shows field visualizations for a numeric field.

- These examples include: *Combo Box, Listbox, SpinEdit, Radio Button Set* and *TrackBar*.
- *SpinEdit* has an *Increment* property.
- *Trackbar* has a *MinimumValue* and *MaximumValue* properties.



The next example includes field visualizations for:

- A *Radio Button Set* and a *Combo Box* for field Gender.
- *Checkbox* and *Combo Box* for On Leave?
- *Image and Text* picklist for Job Status
- *Multi-Line Editbox* for Memo Notes (a string field)

• A *Date Prompt* for a Date Time field:



Advanced field visualization options include:

- A *Dynamic Picklist* which is populated by a Reusable part.
- An *Autocomplete* input box, also populated by reusable part.
- A *Prompt Form* can also be attached to the field, which is linked to a prompt button and which can support simple or complex searches as required.

# Objectives

This exercise provides examples to enhance some of the applications built in previous exercises. More field visualization examples are provided Field Visualization Development in the *LANSA Development Guide*.

To achieve the objectives you will complete these steps:

Step 1. Define a Picklist for iiiGRADE.

Step 2. Define a Dynamic Picklist for SKILCODE

Step 3. Link Dynamic Picklists

Step 4. Implement Dynamic Picklists in By Location filter

Step 5. Create an AutoComplete Visualization for Surname Summary

### **Before You Begin**

Complete exercises VFW020, VFW030, VFW042 and VFW084.

# Step 1. Define a Picklist for iiiGRADE.

- 1. Copy field GRADE to create a new field iiiGrade, select the options to copy rules, visualization and help text. Open the new field in the editor.
- 2. Select the *Visualization* tab, which should contain a *VisualEdit* definition:



3. Add a *Static Picklist* using the toolbar button:



4. Select the *Picklist* component and define the following picklist values:

#### **Caption** Value Default

Pass	Р	Yes
Fail	F	
Merit	Μ	
Distinction	D	

5. Select the *Definition* tab and change the field *Default* value to '**P**', to correspond with the *Static Picklist* default value. See later in the exercise for more detail on this step.

6. On the *Visualization* tab, select the *VisualPicklist* component. Note that it has defaulted to a suitable design, a *Dropdown*.

Widen the visualization slightly so that the word "Distinction" would be shown.

If necessary, reduce the height to the minimum possible.

**Note:** Always make visualizations use the minimum space on the interface.

Select the *Details* tab and change the new visualization to *Default Visual* = **True.** 

- 7. Save your field definition.
- 8. Open iiiVFW19 Employee Skill Detailer in the editor and select the *Source* tab. Position the cursor at the top of the code.
  - a. Display the *Replace* dialog (*Ctrl+H*) and set it up to replace #GRADE with #iiiGrade. Do not *Replace All*.

Replace	wild, 25 Sectors	Barrent of	X
Find what:	#GRADE		<u>F</u> ind Next
Replace with:	#iiiGrade	*	Replace
Match whole we Match case	ord only	(	Replace <u>A</u> ll
Wrap		[	Cancel

b. Use *Find Next*, which will be this line:

Define_Com Class(#GRADE.Visual) Name(#GRADE) Displayposition(2) Hei

b. Use *Replace* to change both entries on this line. Your code should look like the following:

Define_Com Class(#iiiGRADE.Visual) Name(#iiiGRADE) Displayposition(2

c. You should now be positioned on this line:

Define_Com Class(#PRIM_FWLI) Name(#FWLI_7) Manage(#GRADE) Pare

d. Replace #GRADE and *Cancel* the *Replace* dialog.

9. You will now add code to set up iiiGrade from the real field GRADE.

a. At the top of the PHBN_SAVE.Click event routine add this code:#GRADE := #iiiGrade

b. In the uShow method routine, add this code immediately after the FETCH#iiiGrade := #GRADE

c. On the *Design* tab check the appearance of iiiGrade visualization and increase its width if necessary.

Skill Code : : : : : : : :	ABCDEFGHIJ	
Grade Obtained for S	kiļl:::[	-

- 10. Recompile iiiVFW19 Employee Skill Detailer.
- 11. Execute your Framework and display the *Skills 2* command handler for an employee. Note that the Grade obtained for the skill field is now visualized as a dropdown.

🍠 Employee	: Skills 2	(A0193-SMITHSON I	RED)		
Details	🏟 Brief No	tes   🧐 Image   📎 N	lotes 🥬 Skills 2	🦫 Skills	
Code	Grade	Comments	Date Acqu	Skill Code	MADVETI
ADMIN1	D	Met requirement	25/03/98	Skill Code	MARNETI
COM	D		4/05/98		
CS	P		5/05/98	Grade Obtained for Skill	Distinction
HIS	F		2/05/98		
INTRO	P	Met requirement	5/02/98	Comment on skills acquired	Metrequirement
KEY	P	Met requirement	5/02/98	comment on ana acquirea	Piece regariemente
MANAGE1	D	Met requirement	15/03/98	Date Skill Acquired	25/03/08
MARKET1	D	Metreguirement	25/03/98	Date Star Acquired	23/03/30
MARKET2	P		1/05/98		
NOTH	nem	LEEOGBABUK	3/05/98		

#### Note:

- The *Static Picklist* will be automatically positioned to show the field's current value. If the field contains any value not in the picklist, such as blanks, the picklist will show its default caption.
- The field's value is only populated or changed by the visualization when an option is selected in the picklist (visualized as a dropdown in this case). This means if no selection is made, the fields value could be invalid. Defining the field's *Default value* to be the same as the picklist's default avoids this problem.

# Step 2. Define a Dynamic Picklist for SKILCODE

Fields may have *Dynamic Visualizations* defined. Visualizations may include:

- A reusable part which includes logic to *Autocomplete* the value of a field as an entry is typed.
- A *Dynamic Picklist* may be defined with entries retrieved from a file at run time.

For full details refer to the Visual LANSA Developer Guide.

In this example you will create a *Dynamic Picklist* for a new field iiiSkillCode which populates a dropdown from the file SKLTAB.

- 1. Copy field SKILCODE to create a new field iiiSkillCode, copy rules, visualizations and help text. Open the new field in the editor.
- 2. A *Dynamic Picklist* is implemented via a simple reusable part.

Create a new Reusable Part / Dynamic Picklist:

Name: iiiVFW20

### Description: Skill Code Dynamic Picklist

3. A *Dynamic Picklist* RP is a non visual component (ancestor PRIM_OBJT) which implements #prim_dc.iDynamicPicklist.

Implementing *iDynamicPicklist* allows a visualization reusable to *Load* the picklist at runtime.

**Load** - The load method is executed during initialization and any time a monitored value or context changes. This picklist instance is received via the Picklist map. The instance of the picklist is maintained at runtime, meaning that the user must clear the list if new data is required.

See the *Visual LANSA Developer Guide* for more detailed information about *Dynamic Visualizations*.

4. The *Load* method routine which will populate the picklist from the file SKLTAB when the field component is loaded onto a form or panel, needs to be completed.

The method routine redefines the *Load* method in the *prim_dc.iDynamicPicklist* component.

Add the following code, after the comment line **Populate the picklist*, replacing the supplied code provided

```
* Populate the picklist
Select Fields(#skilcode #skildesc) From_File(skltab)
#Picklist.Add( #skilcode #skildesc )
Endselect
```

### Note:

The *Load* method displays a *Define_Map*. This is part of the redefined *Load* method of *prim_dc.iDynamicPicklist*. The parameter passes a reference to a component PRIM_PKLT, named **Picklist** in the Load method.

- The *Picklist.Add* method enables your reusable part to add entries to the picklist.
- The *Picklist.RemoveAll* clears the picklist.
- The *Load* method includes code to clear the picklist.
- Always use *F2 Feature Help* to investigate the definition of shipped components such as PRIM_PKLT.
- 5. Compile your new reusable part.
- 6. The new field iiiSkillCode which you just created, should be open in the editor.
- 7. Select the *Visualizations* tab and use the toolbar button to insert a *New Dynamic Picklist*:



A *Repository Find* dialog is displayed. Select your reusable part iiiVFW20.

- 8. Select the *VisualPicklist* component and increase its width.
- 9. Select the *VisualPicklist* and use the *Details* tab to change its *DefaultVisual* property to **True**.
- 10. Examine the field source code. Note the *DynamicPicklist* component

definition:

```
Begin_Com Role(*Dynamic_Picklist #IIIVFW20) Name(#DynamicPicklist) N
End_Com
```

As with all picklists, dynamic or static, the *NoMatchAction* property needs to be set to determine what to do in the event of the underlying field value not being found in the picklist. For most situations, **ShowValue** is probably the best choice.

- 11. Select the *DynamicPicklist* item. On the *Details* tab, change the *NoMatchAction* property to **ShowValue**.
- 12. Save the field definition.
- 13. Open the *Employee Skill Detailer* (iiiVFW19) in the editor. You will be making similar change as for field iiiGRADE
  - a. Change the SKILCODE component definition to use the dynamic picklist, *iiiSkilcode.VisualPicklist*.

Define_Com Class(#iiiSkillCode.VisualPicklist) Name(#iiiSkillCode) Displa

b. Change the *Flow Manager* item for SKILCODE, to manage iiiSkillCode Define_Com Class(#PRIM_FWLI) Name(#FWLI_10) Manage(**#iiiSkillCode**)

- c. Close the *Replace* dialog.
- d. At the top of the *PHBN_SAVE.Click* event routine add the following code:

#SKILCODE := #iiiSkillCode

e. In the *uShow* method routine, add the following code immediately after the FETCH:

```
#iiiSkillCode := #SKILCODE
```

- f. Switch to the *Design* tab, and increase the width of Skill Code if necessary.
- g. Compile your *Employee Skill Detailer* reusable part (iiiVFW19).
- 14. Execute your Framework and ensure that the correct Skill Code description is shown when an employee skill is selected:

Employee : S	SKIIIS Z (AUT93-	SMITHSON FREDJ			
Details 🥵	Brief Notes  🥹	Image Notes bill	s 2 🛛 🏷 Skills		
Code	Grade	Comments	Date Acquired	Skill Code	Company Introduction
ADMIN1	D	Met requirement	25/03/98	Skir Code	company and oddcaon
COM	D		4/05/98		
CS	P		5/05/98	Grade Obtained for Skill	Pass *
HIS	F		2/05/98		
INTRO	P	Met requirement	5/02/98	Comment on skile acquired	Met remirement
KEY	P	Met requirement	5/02/98	comment on skills acquired	Metrequiement
MANAGE1	D	Met requirement	15/03/98	Data Skill Acquired	5/02/08
MARKET1	D	Met requirement	25/03/98	Date Skill Acquired	5/02/90
MARKET2	P		1/05/98		
OTH	P	GEOGRAPHY	3/05/98		

- 15. Create a new employee skill, using the context menu on the employee skills list view. Notice that the skill code dropdown and grade, initially shows their last value. This is because the fields IIISKILCODE and IIIGRADE are not currently being initialized in the *uNew* method routine.
- 16. Add code to *uNew* to set the new fields to their default value and re-test. Notice that iiiGrade defaults to "Pass" as expected and iiiSkilCode shows blanks, which is its default value, copied from field definition SKILCODE.

# Step 3. Link Dynamic Picklists

The next example demonstrates how dynamic picklists can be linked. A dynamic picklist for Sections can be loaded from the Section table (SECTAB), but the Sections dynamic picklist, must be rebuilt whenever the department changes.

1. Create a new Reusable Part / Dynamic Picklist:

Name: iiiVFW21

### Description: Departments Dynamic Picklist

2. Add the following immediately after the comment line * populate dynamic picklist, replacing the supplied code:

* Populate dynamic picklist Select Fields(#deptment #deptdesc) From_File(deptab) #Picklist.Add( #deptment #deptdesc ) Endselect

Having completed *Step 2- Define a Dynamic Picklist for SkillCode*, you should now be familiar with what this code is doing.

- 3. Compile your new reusable part.
- 4. Create a new field iiiDepartment by copying field DEPTMENT, copy rules, visualizations and help text. Open the new field in the editor.
- 5. Select the *Visualizations* tab. Use the **I Tool** *bar button* to add a *New Dynamic Picklist*. In the Find dialog, select the RP, iiiVFW21.
  - a. Select the new *VisualPicklist* component and adjust the width of the dropdown to display the department description.
  - b. Select the *VisualPicklist* component and use the *Details* tab to change its *DefaultVisual* property to **True**.
  - c. Select the *DynamicPicklist* component and use the *Details* tab o change its *NoMatchAction* property to **ShowValue**.

Details		Definition   Rules and Trigger
DynamicPicklist	- 🕞	🗙 🚥 🗄 🗐 🔹 🚥
Properties Events Metho	ds	Field Parts
🗆 • 🤋		4 🥐 Visualizations
ComponentClassNam	DynamicPicklist	I VisualEdit
ComponentPatternNa	IIIDEPART	VisualPicklist
ComponentTag		Picklists
ComponentTypeNam	IIIDEPART.DynamicPicklist	DynamicPicklist
ProfaultPicklist	False	
P Name	DynamicPicklist	
NoMatchAction	ShowValue	
Owner	#IIIDEPART	

- 6. Save and close the field definition.
- 7. Create a new Reusable Part / Dynamic Picklist:

### Name: iiiVFW22

### Description: Section Dynamic Picklist

8. Add the following code, immediately after the ***Populate the picklist** comment line, replacing the default code provided:

Select Fields(#section #secdesc) From_File(sectab) With_Key(#deptment) #Picklist.Add( #section #secdesc ) Endselect

9. There is a new requirement for the Sections Dynamic Picklist. It needs to monitor changes in the field **DEPTMENT**.

To do this it needs to also implement *Prim_dc.iMonitorSubject* and include a method routine for *ApplyMonitorValue*.

10. Extend your code based on the changes highlighted in red below:

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #PRIM_OBJT *implements #Prim_dc.iDynam Mthroutine Name(Load) Options(*redefine) #Picklist.RemoveAll * Populate the picklist Select Fields(#section #secdesc) From_File(Sectab) With_Key(#Deptment) #Picklist.Add( #Section #Secdesc ) Endselect Endroutine

#### Mthroutine Name(ApplyMonitoredValue) Options(*Redefine) #Deptment := #MonitorSubject.GetValue Endroutine End Com

Note: The *ApplyMonitoredValue* routine displays the *Define_map* defined in its ancestor.

- 11. Compile the new reusable part.
- 12. Create a new field **iiiSection** by copying field SECTION, copy rules, visualizations and help text. Open the new field in the editor.
- 13. Select the *Visualizations* tab and add a *New Dynamic Picklist* and select iiiVFW22.
  - a. Adjust the width of the *VisualPicklist* dropdown.
  - b. Make the *ViusalPicklist* the field's *Default* visualization.
  - c. Select the *DynamicPicklist* component and use the *Details* tab to change the *NoMatchAction* to **ShowValue**.
- 14. Save and close the field definition.

# Step 4. Implement Dynamic Picklists in By Location filter

- 1. Open the *Employees by Location* filter reusable part (iiiVFW05) in the editor.
  - a. In the *Design* view, delete the fields DEPTMENT and SECTION.
  - b. Add fields iiiDepartment and iiiSection
  - c. Adjust the main panel width to show the dropdowns.
  - d. Switch to the Source tab. Position at the top of the code. Use the *Replace* dialog to replace DEPTMENT with iiiDepartment in the following lines:

Group_by Name(#XG_KEYS)...

Def_Cond . .

Evtroutine Handling(#DEPTMENT.Changed)..

e. Replace SECTION with iiiSection, in the following line:

Group_by Name(#XG_KEYS) . .

### Note:

- In the Section's Dynamic Picklist, a monitor is defined with a source of Deptment and a target of Section.
- When iiiDepartment changes, the *ApplyMonitorValue* method in the iiiSection visualization is run. A reference to the source object is received allowing the *GetValue* method to be called to obtain the value.
- Immediately after the *ApplyMonitorValue* has finished, the *Load* method will be run.
- 2. In the filter, iiiVFW05, after the *BEGIN_COM*, define the monitor component, with a source of iiiDepartment and a target of iiiSection.

Define_Com Class(#prim_lm) Name(#DepartmentSection) Source(#iiiDepartn

Changes to department code will be processed by the *ApplyMonitoredValue* method in the *Sections Dynamic Picklist* component (iiiVFW22).

3. Extend the *iiiDepartment.Changed* event routine for the field iiiSection. Add the following code (highlighted in red).

Evtroutine Handling(#iiiDepartment.Changed) Options(*NOCLEARMESSAC Set Com(#Search_Button) Enabled(*SearchOK)

```
* Ensure the Section is valid after a department change
#iiiSection := *null
Select Fields(#SECTION) From_File(sectab) With_Key(#iiiDepartment)
#iiiSection := #SECTION
Leave
Endselect
Endroutine
```

**Note:** The new code ensures the filter changes the value of iiiSection to the first appropriate value, by reading the first record from SECTAB for the selected Department code.

- 5. Compile the *By Location* filter iiiVFW05.
- 6. Execute the Framework and select the *By Location* filter for Employees.
  - a. The sections dropdown should be rebuilt whenever the department dropdown selection is changed.
  - b. When department selection is changed, the first section for that department should be displayed.

the second second second second		x				x
Sy Name 😥 By Locatio	n		5 I M	2 O S C 3 🙆 🖌		
🗸 Clear List		Search	Employee	Description	Departm	Salary Star
Department Code	Fleet Administration		A1016	Turner Jack	FLT	22,000
Section Code	Internal Admin					
			•			- F

# Step 5. Create an AutoComplete Visualization for Surname

An *AutoComplete* visualization completes the input box as you type. The reusable part can return a value based on whatever logic is required. In this case the Employee file will be read using a logical file in surname order, using *Generic(*YES)*. The first match returns the surname.

Having defined the Surname *Autocomplete* visualization, recompiling the By Name filter, iiiVFW04, will demonstrate its implementation.

- 1. Create the field iiiSurname by copying field SURNAME and copy rules, visualizations and help text. Open the new field in the editor. You will complete this field definition in a later step.
- 2. Create a new Reusable Part / Panel:

### Name: iiiVFW23

### Description: Surname AutoComplete

- 3. Replace the code with the source provided in VFW090 Appendix A.
- 4. Replace all occurrences of <FIELD> with #iiiSurname, where **iii** = your initials.
- 5. Compile the new reusable part. Switch to the *Design* view.

Note that the panel, has an *Attachment Manager* and contains field iiiSURNAME, with a *MarginLeft* property of **0**.

If necessary re-size your panel as shown:



- 6. Review the code provided:
- The *Begin_Com* extends PRIM_PANL and implements *prim_dc.iMonitorSubject*.

Begin_Com Role(*EXTENDS #PRIM_PANL *implements #Prim_dc.iMonito

*iMonitorSubject* is the simplest and most common form of *Visual Host*. All visual field instances implement this interface. It has two methods that will be called depending on how the field is used.

*ApplyMonitoredValue* is typically called when the field is used as the target

of a monitor. A reference to the Monitor Source component is received in the *iMonitorSubject* input map.

*GetValue* can be called to obtain the value of the field. As *GetValue* applies to any field type, the result map is a variant.

The *ValueChanged* event can be signaled to indicate that the value of the variable has changed. This equates to the Changed event.

- The *Visual Host* has a property 'Value' which passes and receives the value of SURNAME
- The *KeyPress* event routine checks if a character key was pressed and if the field is full
- The *PrepareAutoComplete* method ensures that the selected characters run left to right.
- The *CanAutoComplete* method tests if the selection starts at the end of the current value
- The *Autocomplete* method invokes the *GetCandidate* method which reads the next record from the employee file (LF PSLMST2) generically.
- The *KeyPress* event signals *ValueChanged*. This event is defined in *Prim_dc.iMonitorSubject*.
- 7. Switch to the field iiiSurname which should be open in the editor. Use the *Reusable Type Toolbar* button to insert a *New Visual Host*.



- 8. In the *Repository Find* dialog, select the RP, iiiVFW23 and click *OK*.
- 9. Select the *VisualHost* and use the *Details* tab to change the *DefaultVisual* to **True**.
- 10. Make the *VisualHost* wide enough to show the full iiiSurname field.
- 11. Save the field definition.
- 12. Open the *By Name* Filter (iiiVFW04) in the editor.
- 13. On the *Design* tab, delete the field SURNAME and add field iiiSurname.

a. Check that the field definition uses *VisualHost* and change it to this component if necessary.

Define_Com Class(#iiiSurname.VisualHost) Name(#iiiSurname) Displayposit

- b. Change the *LabelPosition* property to **Top**
- c. Change the *LabelHorAlignment* to **Left**
- 14. Select the filter's left hand panel (BODY_PANEL) and select the *Layout Helper*. If necessary, open the *Layout Helper* from the *Home* ribbon, *Views* menu.

Layout Helper	Design Source Repository Details
Layout Managed Component	IIIVEW04 - Employee Filter by N
BODY_PANEL -	
Layout 🖆 🗙	Clear List
BODY_FLOW *	Employéé Surname,
Children 👔 🗙 👔 🗙 Child Details As Ch	aAbBcCdDeEfFgGhHII)J
VICLEAR LIST	
Category Flow Rules	

15. In the *Children* list, select iiiSURNAME so it is managed by the BODY_FLOW manager.

]	IIIVFW04 - Employee Filter by Name																×													
Clear List											•	•	[		-	S	Search													
	É	'n	ble	oiy	ė	ė	S	uir	'n	ar	ne		•	•	•	•	:	:	•	• • •	•	•	•	•	•	•	•	•	•	:
ŀ	a	A	Ы	Bo	C	d	De	E	ff	9	G	hł	li	IJ.	נ		•	:		•	•		:		•	:	•	:	•	
	•	:	:	•	:	:	•	:	•	:	:	:	:	:	•	•	:	:	:	•	:	:	:	:	:	:	:	:	:	:
	•••••••••••••••••••••••••••••••••••••••	•	•	•	:	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	:	•	:	:
ļ	:	:	:	•	:	:	:	:			:	:	:	:	:	:	:	:	-	:	:	:	:	:	:	:	:	:	:	:

16.Switch to the Source tab. Change the following lines of code to use your field iiiSURNAME as shown:

Group_By Name(#XG_Keys) Fields(#iiiSurname) Def_Cond Name(*SearchOK) Cond((#iiiSurname *NE *Blanks)) #avFrameworkManager.avRestoreValue Withid1(*Component) Withid2(#iiiSURNAME.Name) Toavalue(#iiiSurname) #avFrameworkManager.avSaveValue Withid1(*Component) Withid2(#iiiSURNAME.Name) Fromavalue(#iiiSurname) Evtroutine Handling(#iiiSurname.Changed) Options(*NOCLEARMESSAGES *NOCLEARERRORS)

17.Add the highlighted code shown to the Search_Button.Click event routine:

```
Evtroutine Handling(#Search_Button.Click)
Options(*NOCLEARMESSAGES *NOCLEARERRORS)
#Com_Owner.uSelectData
#surname := #iiiSurname
Endroutine
```

18. Compile the filter.

19. Execute the Framework and test the *By Name* filter for employees.

Type S into Surname.

**Note:** the *AutoComplete* visualization returns the first matching full surname, but selects text at the right hand side, except for your typed value (S).



This ensures the selected text will be replaced, if you continue typing.

Clear the text except "S" and click *Search* to see all employees with names starting "S", in name order.

Typing SM, SMY or SN, SNA or SNE will enable you to see that the *Autocomplete* logic refines the returned values by repeating the SELECT each time the search value changes.
## Summary

#### **Important Observations**

• Field Visualizations are ideally suited to static lists of data for example, Yes, No; Male, Female and so on.

## Tips & Techniques

- The field visualization is controlled by the visualization class. *BEGIN_COM ROLE(*Visual <class name>)*...
- For fields used stand alone on forms, the following classes can be used:
  - Edit Box (PRIM_EVEF)
  - Spin Edit (PRIM_EVSE)
  - Button Sets (PRIM_EVPL)
  - Combo Boxes (PRIM_EVPL)
  - Image Sets (PRIM_EVPL)
  - Check Boxes (PRIM_EVPL)
  - Track Bars (PRIM_EVTB)
  - Progress Bars (PRIM_EVPB)
- The default picklist visualization is a set of radio buttons for the PRIM_EVPL visualization class, that is, the default is *Appearance(ButtonSet)*.
- For the picklist, the *BEGIN_COM Role(*Visual #PRIM_EVPL) Name(#VisualPicklist) Appearance(xxxxxxxx)* may have values where xxxxxxxxx is *ButtonSet*, *CheckBox*, *ListBox*, *DropDown*, *Image*, and *ImageAndText*.
- When using a check box, the order of the picklist values is important. Clicking the check box will set the corresponding value for the field from the picklist items. Values are chosen from the following rule.
  - Checkbox Unchecked = First picklist item value
  - CheckBox Checked = Last picklist item value
  - CheckBox grayed = Any picklist item between first and last or an invalid value. (for example, Changing the value of the field to a value that is not the first or last item in the picklist will result in the checkbox being

grayed.)

- A single field may have many field visualizations. Field visualizations can improve developer productivity and improve the consistency of your applications.
- When you change an existing field visualization, forms and reusable parts which use it will not change until they are recompiled.

# What I Should Know

- How to insert a new field visualization.
- How to change a field visualization by changing the visualization class.
- The purpose of the *Appearance()* property when defining a visualization.
- How to change a field visualization by changing the component properties.
- How to create a picklist visualization.
- How to select a field visualization for a form.

# VFW090 – Appendix A

Use the following code to create iiiVFW21 – Surname Autocomplete Function Options(*DIRECT) Begin_Com Role(*EXTENDS #PRIM_PANL *implements #Prim_dc.iMonito Define_Com Class(#PRIM_ATLM) Name(#ATLM_1) Define_Com Class(#PRIM_ATLI) Name(#ATLI_1) Attachment(Center) Paren Define Com Class(<FIELD>.VisualEdit) Name(<FIELD>) Displayposition(1) Define Com Class(#PRIM ATLI) Name(#ATLI 2) Attachment(Center) Paren Define_Com Class(#PRIM_ATLI) Name(#ATLI_3) Attachment(Center) Mana Define_Pty Name(Value) Get(GetPropertyValue) Set(SetPropertyValue) Ptyroutine Name(GetPropertyValue) Define Map For(*Output) Class(#prim alph) Name(#Property) #Property := <FIELD> Endroutine Ptyroutine Name(SetPropertyValue) Define_Map For(*Input) Class(#prim_alph) Name(#Property) <FIELD> := #Property Endroutine Mthroutine Name(ApplymonitoredValue) Options(*redefine) * No redefinition required Endroutine Mthroutine Name(GetValue) Options(*redefine) * No redefinition required Endroutine Evtroutine Handling(<FIELD>.KeyPress) Handled(#Handled) Keycode(#Keyt * If the field isn't full If (<FIELD>.CurSize <> <FIELD>.FieldLength) * If a character entered If (#KeyCode = isChar) #Com_Owner.PrepareAutoComplete If (#Com_owner.CanAutoComplete) #Handled := True #Com_owner.AutoComplete( #Char ) Signal Event(ValueChanged) Endif Endif Endif

Endroutine

Evtroutine Handling(<FIELD>.Changed)

* Handle all other key presses that might affect the value

```
Signal Event(ValueChanged)
```

Endroutine

Mthroutine Name(CanAutoComplete) Help('Can we autocomplete?') Access(* Define_Map For(*Result) Class(#prim_boln) Name(#Result)

* If selection doesn't start at the end of the value, autocomplete is not appropria #Result := (<FIELD>.SelectionEnd = (<FIELD>.Trim.cursize + 1))

Endroutine

```
Mthroutine Name(AutoComplete) Access(*private)
```

Define_Map For(*Input) Class(#prim_alph) Name(#Char) Help('Character just

Define_Com Class(#prim_nmbr) Name(#Start)

Define_Com Class(#prim_alph) Name(#Candidate)

#Start := <FIELD>.SelectionStart

```
#Candidate := #Com_owner.PrepareCandidate( #Char )
```

```
<FIELD> := #Com_owner.GetCandidate( #Candidate )
```

```
* Set selection to be startposition + 1 to the end
```

```
<FIELD>.SelectionStart := #Start + 1
```

```
<FIELD>.SelectionEnd := <FIELD>.Trim.cursize + 1
```

Endroutine

Mthroutine Name(PrepareAutoComplete) Help('Prepare Selection in the value Define_Com Class(#prim_nmbr) Name(#Transition)

```
* If Start is greater than end, reverse the selection points
```

```
If (<FIELD>.SelectionStart > <FIELD>.SelectionEnd)
```

```
#Transition := <FIELD>.SelectionStart
```

```
<FIELD>.SelectionStart := <FIELD>.SelectionEnd
```

```
<FIELD>.SelectionEnd := #Transition
```

Endif

Endroutine

Mthroutine Name(PrepareCandidate) Help('Prepare the input value ready for lc Define_Map For(*Input) Class(#prim_alph) Name(#Char) Help('Character just Define_Map For(*Result) Class(#Prim_alph) Name(#Result)

```
* If selection is the whole word, only use the char supplied by the event
```

If (<FIELD>.SelectionStart = 1)

#Result := #Char.uppercase

Else

* Get anything to the left of the cursor start position and append the last key pr #Result := (<FIELD>.substring( 1 (<FIELD>.SelectionStart - 1) ).trim + #Chai Endif Endroutine Mthroutine Name(GetCandidate) Access(*private) Define_Map For(*Input) Class(#prim_alph) Name(#Candidate) Define_Map For(*Result) Class(#prim_alph) Name(#Result) * If no record found, the last value entered is still the right answer #Result := #Candidate * Find the first record starting with the candidate value Select Fields(#SURNAME) From_File(pslmst2) With_Key(#Candidate) Genei #Result := #SURNAME Leave Endselect Endroutine End_Com

## VFW100 – Define a Parent/Child Instance List

The shipped VL Framework *HR Demo Application* has a business object *Organisations* which demonstrates how the instance list can be configured as a tree with a number of levels:

Organizations				
x				
🗄 😭 Favorites		Name	Code / Id	Address 1
🖃 🌏 HR Demo Application		ADMINISTRATOR DEPT	ADM	
Organizations		🖃 💼 Sections		
Se Resources	<i>\</i>	INTERNAL ADMIN SRV	01	125 Main St,
🗄 🎒 iii HR Application	444	🖃 🍇 Resources		
🗄 🎒 JI HR Application	M	DOUGLAS, ADAM PETER	A1020	6 Reading Avenue,
🗄 🎒 Marcus Instance Example	0	MRS BRICK, GILL	A1404	22 Moton Street
🗄 🧔 Programming Techniques	100	REDFORD,ROBERT	A1509	122 Arthur Street
🗄 🔩 Administration	3		02	123 Pacific Highway,
		ACCOUNTING SECTION	03	252 Canterbury Road,
	N	SALES & MARKETING	04	121 Pitt Town Road
		MAINTENANCE	05	121 Railway Parade
		PERSONNEL SECTIONXX	06	121 Smith St
				III
	_			
		<u> </u>	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

The standard instance list manager is configured to show 3 business objects. Implementing this example within the HR Demo Application required the following steps:

- Define business objects: Organisations, Sections and Resources.
- Create a hidden filter to populate the instance list with *Organisations*.
- Define Instance List relationship for Organisations to Sections.
- Create a relationship handler function to expand *Organisations* with *Sections*.
- Define the Instance List relationship for Sections to Resources.
- Create relationship handler function expanding *Sections* with *Resources*.

You will find a full description of the topic *Instance Lists with different types of object* in the *Visual LANSA Framework Guide / Framework Programming / List Manager and Instance Lists.* 

## Objectives

For this exercise you will define new Departments and Sections business objects for your iii HR Application. The Departments instance list will then be defined with a child of Sections.

- A Hidden filter will populate the instance list with all departments.
- A Relationship Handler function will add sections to the instance list when a

department is expanded.

	C C			
escription	Code	Address Line 1	Address Line 2	
Administration	ADM			
4 🕡 Sections				
Internal Admin	01	125 Main St,	Blacktown	
Purchasing	02	123 Pacific Highway,	North Sydney	
Accounting	03	252 Canterbury Road,	Canterbury	
Sales & Marketing	04	121 Pitt Town Road	Pitt Town	
Maintenance	05	121 Railway Parade	Woodsville	
Personnel	06	121 Smith St	Newtown	
Vehicle Maintenance	09	121 Smith Street	Newtown	
Internal Auditing	AUD			
Fleet Administration	FLT			
Group Accounts	GAC			

• To meet this objective you will complete the following:

Step 1. Define New Business Objects

Step 2. Create a Hidden Filter for _Departments.

Step 3. Create a Relationship Handler to Load _Sections

Step 4. Access the Properties of Hidden Child Objects Summary

## **Before You Begin**

You should complete exercises VFW020, VFW030 and VFW040.

## **Step 1. Define New Business Objects**

In this step you will create two new business objects and define a number of actions for each, using the *Instant Prototyping Assistant*.

- 1. Open the *Framework as Designer*, and start the *Instant Prototyping Assistant* from the *Framework* menu.
  - a. Define two new business objects: **_Departments** and **_Sections**. Note the underscore which avoids a clash with existing demo application objects.
  - b. Define actions Transfers, Copy, Resources, Images and Picture.
  - c. Attach **Details, Transfers, Copy** and **New** to the **_Departments** business object
  - d. Attach **Details, Resources, Images, Picture, Transfers** and **New** to the **__Sections** business object
  - e. Attach **_Departments** and **_Sections** to the *iii HR Application*.
  - f. Click *Next* then the *Finish* button to update your *iii HR Application*

Your extended Framework should look like the following:



- 2. Save your Framework.
- 3. In this step you will refine the new prototype objects which will be used in a number of the following exercises:
  - a. Open the **_Departments** *Business Object Properties* dialog. Select the *Commands Enabled* tab. Select the *Copy* action and make it a *Business*

**Object** Command.

When the *Please Note.....* dialog appears, unselect the "Warn me...." *Checkbox* and *Close* the dialog. You will save and restart the Framework in a later step.

- b. Select the *Command Display* tab, and change the *Object Command Display* setting to *Separate Stay on top window*.
- c. Select the *Icons* tab and give the _Departments any suitable icon.
- d. Close the Business Object Properties dialog.
- e. Open the Properties dialog for the **_Sections** business object. Select the *Icons* tab and give the **_Sections** business object any suitable icon.
- f. *Save and Restart* the Framework.
- 4. In this step you will define the instance list relationship and additional columns.
  - a. Open the *Properties* dialog for the _Departments business object.
  - b. Select the Instance List / Relationships tab.
  - c. In the list of business objects, select _Sections. Define it as a Child or Descendant.
  - d. Unselect Allow Selection from Navigation Pane.
  - e. Unselect Side by Side Display. Close the Please Note.... dialog.



f. Define the following additional *Instance List* columns in the sequence shown:

#### **Type Column Caption**

AColumn1	Address Line 1
AColumn2	Address Line 2
AColumn3	Address Line 3
NColumn1	Post Code
AColumns4	Phone Number

- g. Change the VISUALID1 column *Caption* to **Code**.
- h. Close the Properties dialog and Save and Exit the Framework.
- i. Execute the Framework as an End User. Click the *Emulate Search* button and expand the Instance List tree view.

Your design should now look like the following:



- 5. Validate your prototype _Departments and _Sections business objects.
- 6. Confirm the Instance list columns and the commands available for the new business objects.
- 7. Close the Framework.

## **Step 2. Create a Hidden Filter for _Departments.**

The instance list will be initially loaded with all departments from the Department table (DEPTAB).

1. Create a new *Reusable Part / Object*:

#### Name: iiiVFW24

#### Description: Departments Hidden Filter

- 2. Give the reusable part an ancestor of VF_AC007.
- 3. Create an *uInitialize* method routine after the BEGIN_COM statement: Mthroutine Name(uInitialize) Options(*Redefine)

Endroutine

4. In the *uInitialize* routine make the filter hidden, so that all that will show at run-time is the instance list:

Set #Com_Owner avHiddenFilter(TRUE)

5. Then indicate that the instance list updating is about to start and clear the instance list:

#avListManager.BeginListUpdate
#avListManager.ClearList

6. Read all the departments and add them to the instance list:

Select Fields(#Deptment #DeptDesc) From_File(DEPTAB) #avListManager.AddtoList Visualid1(#DeptDesc) Visualid2(#Deptment) Akey EndSelect

#### Note:

- It is necessary to initialize the additional columns with the *AddToList* is invoked, as you will later populate the list with entries for _Sections, which will fill these columns.
- The *BusinessObjectType()* parameter must use your business object *User Object Name/Type* and is uppercase.
- 7. Lastly indicate that instance list updating is now complete:

#avListManager.EndListUpdate

Your code should look like this:



- 8. Compile the filter.
- 9. Display the Framework and open the *Properties* dialog for the _Departments business object.
- 10. Display the *Filter / Snap-in Settings* tab.
- 11. Specify iiiVFW24 as the real filter.
- 12. Close the _Departments' properties dialog. *Save and Restart* your Framework.
- 13. Select _Departments so that the hidden filter loads the departments into the instance list.

escription	Code	Address Line 1	
Administration	ADM		
Internal Auditing	AUD		
Fleet Administration	FLT		
Group Accounts	GAC		
Information Services	INF		
Legal	LEG		
Mgmt. Information	MIS		
Marketing	MKT		
Sales & Distribution	SD		
Travel	TDVI		

14. Expand a Department. Notice that no _Sections are loaded. You will create the relationship handler that loads the sections in the next step.

# Step 3. Create a Relationship Handler to Load _Sections

In this step you will create a relationship handler that loads Sections into the instance list when a Department is expanded.

You could have loaded the all the Sections in the hidden filter code together with the Departments, but by using a relationship handler you can improve application performance by first only adding root or parent objects to the instance list and then dynamically adding the child objects.

1. In the Visual LANSA editor, create a new *Process* **iiiVFPR01 – Framework Functions.** 

#### Do not open the Process in the editor.

Create a new *Function* belonging to process iiiVFPR01.

Name: iiiVF01

#### Description: _Sections Relationship Handler

- a. No *Template* is required.
- b. Enable for RDMLX.
- c. Open the Function in the *Editor*.
- 2. Replace the existing code in the function with this code that indicates that this function is a relationship handler:

FUNCTION OPTIONS(*DIRECT *LIGHTUSAGE) RCV_LIST(#VIS_LIST INCLUDE PROCESS(*DIRECT) FUNCTION(VFREL1) INCLUDE PROCESS(*DIRECT) FUNCTION(VFREL2)

The VFREL1 and VFREL2 functions which you have include the standard definitions for relationship builder functions.

3. Start your code after the included functions. Add code to clear all the keys and additional columns in the instance list:

EXECUTE SUBROUTINE(CLEARKEYS) EXECUTE SUBROUTINE(CLEARCOLS)

The subroutines you call in the relationship handler are contained in the VFREL2 function.

4. Get the key value of the selected department:

#DEPTMENT := #SRC_AK1

5. Select the sections in the current department and set the values of the

instance list entry:

SELECT FIELDS(*ALL) FROM_FILE(SECTAB) WITH_KEY(#DEPTMEN EXECUTE SUBROUTINE(SETAKEY) WITH_PARMS(1 #DEPTMENT) EXECUTE SUBROUTINE(SETAKEY) WITH_PARMS(2 #SECTION) EXECUTE SUBROUTINE(SETNCOL) WITH_PARMS(1 #SECPCODE) EXECUTE SUBROUTINE(SETACOL) WITH_PARMS(1 #SECADDR1) EXECUTE SUBROUTINE(SETACOL) WITH_PARMS(2 #SECADDR2) EXECUTE SUBROUTINE(SETACOL) WITH_PARMS(3 #SECADDR3) EXECUTE SUBROUTINE(SETACOL) WITH_PARMS(4 #SECPHBUS) EXECUTE SUBROUTINE(SETACOL) WITH_PARMS(4 #SECPHBUS) EXECUTE SUBROUTINE(ADDTOLIST) WITH_PARMS('_SECTIONS' #SI ENDSELECT

**IMPORTANT:** Ensure the *ADDTOLIST WITH_PARMS()* contains the correct name for your _Sections business object.

- The SETAKEY subroutine sets the alpha key values of the child instance list. The first parameter of the subroutine is the key position and the second parameter is the value of the key. There is also a SETNKEY subroutine to set a numeric key.
- The SETNCOL and SETACOL subroutines add additional columns for the child instance list entry.
- The *ADDTOLIST* subroutine adds the entry to the instance list. The first parameter of the subroutine is the child business object name, the second parameter is the *Visual ID 1* column and the third parameter is the *Visual ID 2* column.

Your code will now look like this:

- 6. Compile the function.
- 7. Open the Framework as Designer.
- 8. Display the *Properties* dialog of the _Departments business object.
- 9. In the Instance List/Relations tab select the _Sections business object.
- 10. In the *Relationship Handler* field, enter the function name **iiiVF01** for the relationship handler.



- 12. Close the _Departments properties dialog.
- 13. Save and Restart the Framework.
- 14. Select the _Departments business object in the iii HR application.
- 15. Expand a department in the instance list.

Description	Code	Address Line 1	Address Line 2
<ul> <li>Administration</li> </ul>	ADM		
4 4 Sections			
Internal Admin	01	125 Main St,	Blacktown
Purchasing	02	123 Pacific Highway,	North Sydney
Accounting	03	252 Canterbury Road,	Canterbury
Sales & Marketing	04	121 Pitt Town Road	Pitt Town
Maintenance	05	121 Railway Parade	Woodsville
Personnel	06	121 Smith St	Newtown
Vehicle Maintenance	09	121 Smith Street	Newtown
Internal Auditing	ALID		

When you expand each department, the sections are loaded dynamically.

**Note:** The instance list displays the additional columns for _Sections which you defined in Step 1.

# Step 4. Access the Properties of Hidden Child Objects

In this step you will learn how to access the properties of the hidden child business object _Sections which is not visible in the navigation pane.

- 1. Execute your Framework as Designer, and open the *Framework* menu and select the *Applications*... menu option.
- 2. Select the iii HR application.
- 3. Select the _Sections business object to display its properties.

(New)	•	5		$\bigcirc$	Quick Find
(Properties)		6		0	
( Applications )	•	0	Administration - Business Objects		
(Commands)		0	Administration - Properties		
(Menus)			Favorites - Business Objects		x
(Design Code Tables)			Favorites - Properties		
(Deserve Cading Assistant )	-		HR Demo Application - Business Objects	•	Address Line 3
(Program Coding Assistant)		4	HR Demo Application - Properties		Mudress Life 5
(Instant Prototyping Assistant)		dy	iii HR Application - Business Objects	•	Departments (III_DEPARTMENT)
	-	dy.	iii HR Application - Properties		Employees (EMPLOYEES)
(RAMP TOOIS )	_	dy.	JI HR Application - Business Objects		Reports (REPORTS)
(Virtual Clipboard)	•	dy.	JI HR Application - Properties		Section Employees (SECTION_EMP
(14 T-1-)		dy.	Marcus Instance Example - Business Objects	•	Sections (III_SECTION)
(Merge Tool )		dy	Marcus Instance Example - Properties		NSW

4. Close the *Properties* dialog of the _Sections business object.

**Note:** There is an alternative way of displaying the properties of child business objects which are not accessible from the navigation pane. That is:

- a. Display the sections in a department in the instance list.
- b. Double-click on a section to display the properties of the _Sections business object.

## Summary

### **Important Observations**

- You can create instance lists that contain more than one type of object. You do this by defining relationships between business objects. The relationships can either be peer-to-peer or parent-child.
- In situations where you want to completely fill the business object instance list programmatically, the filter has no meaningful interaction with the end-user and can be hidden from view.
- A relationship handler is an RDML function that is called to dynamically expand the relationship between a parent and child object. By doing this you can improve filter performance by only adding root or parent objects to the instance list initially.
- The Framework instance list can display up to 10 alphanumeric and/or 10 numeric additional columns in an instance list.

# **Tips & Techniques**

- The shipped framework demo applications contain many examples. The *Advanced* section of the *Programming Techniques* sample application has examples of advanced instance lists.
- LANSA supplies a sample relationship handler to copy from when you create your relationships. The source is stored in function DF_REL01 in the process DF_PROC.

## What I Should Know

- How to create a parent-child relationship between business object.
- How to create a hidden filter.
- How to write a relationship handler.
- How to add additional columns to the instance list.

# VFW102 – Field Visualizations in a Grid

- Grid, List View and Tree Views can have columns with field visualizations.
- In a list component, the properties of the column controls the visualization.
- When used in a list column your field definition doesn't necessarily need a visualization defined.
- The field's picklist definition may be used in a list column.

🛄 Secti	Section : Employees (INTERNAL ADMIN SRV-01)							
Deta	ls 🔒 Employees 🔃 🔟 Images 🛛 🗔 Pic	ture 🛛 🤆 Trans	ifer					
Code	Full Name	Start Date	Salary	Tax Paid	On Leave?	Status	Memo Notes	
A1002	SMYTHE, JOHN	01/01/1977	25,000.04	High		Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.	
A1005	SMITHS, PETER	01/02/1971	46,700.04	Average		Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.	
A1020	DOUGLAS, ADAM PETER	01/02/1988	121,500.04	Average		Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.	
A1404	MRS BRICK, GILL	01/05/1994	12,345.04	Average	V	Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.	
A1509	REDFORD, ROBERT	19/02/1995	10.00	Average	V	Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.	

# Objectives

In this exercise you will create a Resources for Section command handler, displaying a Grid with a number of columns using visualizations.

To achieve these objectives you will complete the following:

Step 1. Define New Fields

Step 2. Create the Resources for Section Command Handler

Step 3. Create a Prompt Form for Employee Number

Summary

## **Before You Begin**

Complete exercises VFW030, VFW040, VFW042 and VFW102.

# Step 1. Define New Fields

In this step you will define three new fields for employees: On Leave, Status and Memo Notes.

You will also create a picklist for salary.

1. Create the following fields:

Name	Description	Туре	Length	Default Value
iiiOnLeave	Employee On Leave Flag	Alpha	1	Ν
iiiStatus	Employee Memo Status	Alpha	3	NML
iiiMemo	Employee Notes	String	512	

- 2. Create field iiiEMPNO by copying field EMPNO. Copy rules, visualizations and help text. No changes to iiiEMPNO are required at this stage.
- 3. Create field iiiSALARY by copying field SALARY. Copy rules, visualizations and help text.
- 4. Create field iiiTAXPAID by copying field SALARY. Do not copy rules, visualizations or help text.
- 5. Open field iiiTAXPAID in the editor and create add a *Static Picklist* and define the following values:

Caption	Value
Small	1
Medium	1000
Large	5000
Very Large	10000

6. Save your changes.

7. Define a Static Picklist for field Employee Memo Status, which includes image components:

Image	Caption	Value	Default
xImageFavorites16	Normal	NML	Yes
xImagePause16	Urgent	URG	
xImageNew16	Late	LTE	
xImageOpen	Very Late	VLT	

Note that the *VisualPicklist* component will default to a DropDownList. Only the Picklist component will be used by the Grid component. To use this visualization on a form or panel, you would set the *VisualChecklist* to **Image** or **Image and Text**.

## Step 2. Create the Resources for Section Command Handler

1. Create a new *Reusable Part / Panel*:

Name: iiiVFW25

#### Description: Resources for Section

- 2. Give the reusable part an ancestor of VF_AC010.
- 3. Use the *Design* ribbon, to give the reusable part an *Attachment manager*.
- 4. Drop a Grid component into the center of the panel, so that it is attached to all four sides.
- 5. Define columns in the Grid using the following fields and set the *Caption* property of each column as shown in the table:

Field	Caption
iiiEMPNO	Code
FULLNAME	Full Name
STD_DATEX	Start Date
iiiSALARY	Salary
iiiTAXPAID	Tax Paid
iiiONLEAVE	On Leave?
iiiSTATUS	Memo Status
iiiMEMO	Notes

- 6. Change *Caption Type* of each column to **Caption**.
- 7. Change the *CaptionAlign* for each column to **Left**.
- 8. Change the *ReadOnly* property for all columns, except FULLNAME, to **False**.
- 9. Change other column properties as shown in the following table:

Field	Property	Value

iiiSALARY	EditAppearance	SpinEdit	
	DisplayAppearance	Edit	
iiiTAXPAID	EditAppearance	Edit	
	Display Appearance	Default	
	UsePicklist	True	
iiiONLEAVE	DisplayAppearance	CheckBox	
	EditAppearance	CheckBox	
iiiSTATUS	EditAppearance	Image and Text	
	DisplayAppearance	Image and Text	
	UsePicklist	True	
iiiMEMO	DisplayAppearance	MultiLineEdit	
	EditAppearance	MultiLineEdit	

10. Open the field iiiONLEAVE in the editor and add a *Static Picklist* and define the following values:

Caption	Value
In the Office	Ν
On Leave	Y

- 11. Save your changes.
- 13. In the reusable part IIIVFW25, open the *Design* view, select the grid columns for iiiTAXPAID and then iiiONLEAVE and change the *UsePicklist* property to **True**.
- 14. Save your changes.
- 15. Select the **Grid** and change the *RowSizing* property to **ContentHeight**.

16. Select the *Source* tab and create an *uExecute* method routine, which redefines the ancestor method:

```
Mthroutine Name(uExecute) Options(*redefine)
Endroutine
```

17. Add the following code to complete the *uExecute* routine:

```
#com_ancestor.uExecute
#avlistmanager.getCurrentInstance Akey1(#deptment) Akey2(#section)
Clr_List Named(#GRID_1)
#iiiONLEAVE := N
#iiiSTATUS := NML
#iiiTAXPAID := 1000
#iiiMEMO := 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquar
Select Fields(#empno #surname #givename #salary #startdter) From_File(pslm
#iiiEMPNO := #EMPNO
#iiiSALARY := #SALARY
#std_datex := #startdter.asdate( YYMMDD )
#fullname := #surname + ', ' + #givename
Add_Entry To_List(#GRID_1)
Endselect
```

18. Review this code which:

- Invokes the ancestor *uExecute* method
- Invokes the list manager and retrieves the keys for the current entry
- Clears the Grid
- Sets up fixed values in a number of fields
- Sets STD_DATEX to a date based on the field STARTDTER (a signed numeric, length 6) in form YYMMDD.
- Sets FULLNAME value based on SURNAME and GIVENAME
- Changes IIIEMPNO to EMPNO
- Changes IIISALARY to SALARY
- Adds an entry to Grid
- 19. Compile reusable part iiiVFW25.
- 20. Execute the Framework as a Designer.

- 21. Open the _Sections properties dialog by expanding a department in the instance list and double clicking on a section.
- 22. Select the *Commands Enabled* tab, select the *Resources* action and plug in the command handler iiiVFW25.
- 23. Save and Restart the Framework.
- 24. Select the Employees command handler for a section. Your design should look like the following:

Deta	ils 🚺 Images	Picture	Resources	🕛 🍤 Transf	fers		
Code	Full Name	Start Date	Salary	Tax Paid	On Leave?	Memo Status	Notes
A1001	Jones, Shirley	01/02/1988	10,000.00	0		Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.
1012	Paul, Patrick	01/05/1986	10,000.00	0	<b>[</b> ]	Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.
A1013	Pattinson, George	01/12/1985	10,000.00	0		Normal	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida suscipit nulla, non tincidunt est ultrices nec.

The height of each row is based on the right hand column, due to the *RowSizing* property of **ContentHeight** for the grid. *Content Height* is taken from the right hand column.

- 25. When you click on the columns to edit a value, check that the behavior is as per the following list:
  - Start Date displays a date prompt when you click on the prompt button in the cell.
  - The Salary value displays the actual value and may be changed using its SpinEdit buttons.
  - Tax Paid displays the picklist values in a dropdown when you edit its value. The value in display mode is based on the picklist value.
  - On Leave is always displayed or edited as a checkbox.
  - Status display the image and text from the picklist. When you click in this column the value rotates through the picklist values.
  - The Memo Notes column value can be extended by typing in additional text. When you position into another cell, the column height is adjusted.

## Step 3. Create a Prompt Form for Employee Number

In this step you will create an employee prompt form, change the iiiEMPNO field definition and configure the employee number column to use the prompter form.

1. Create a new *Form / Basic Form*:

#### Name: iiiVFW26

#### Description: Employee Prompt

The prompter form will display a list of all employees and return Employee number for the selected employee.

The list entry is highlighted if a valid employee number was entered in the input field.

2. Copy and paste the following code to replace the new form's initial code. Ignore errors initially. The form uses a reusable part which you will create in the next step.

Function Options(*DIRECT)

BEGIN_COM ROLE(*EXTENDS #PRIM_FORM) DEFAULTPTY(P_EMPL

DEFINE_COM CLASS(#PRIM_GPBX) NAME(#GPBX_1) CAPTION('All E

DEFINE_COM CLASS(#PRIM_LTVW) NAME(#LISTVIEW) COLUMNBU

DEFINE_COM CLASS(#PRIM_LVCL) NAME(#LVCL_3) DISPLAYPOSIT

DEFINE_COM CLASS(#PRIM_ATLM) NAME(#LAYOUT1)

DEFINE_COM CLASS(#PRIM_ATLI) NAME(#IL301) ATTACHMENT(Cen

DEFINE_COM CLASS(#PRIM_PANL) NAME(#PANL_1) DISPLAYPOSIT

DEFINE_COM CLASS(#PRIM_ATLI) NAME(#ATLI_1) ATTACHMENT(B

DEFINE_COM CLASS(#PRIM_PHBN) NAME(#PB_OK) BUTTONDEFAU

DEFINE_COM CLASS(#PRIM_LVCL) NAME(#LVCL_1) DISPLAYPOSIT]

Define_Pty Name(P_EMPLOYEE_NUMBER) Get(get_empno) Set(SET_EM

DEFINE_COM CLASS(#iiiVFW26) NAME(#Collections) scope(*Application

```
* search and Set Focus on first employee entry
Ptyroutine Name(SET_EMPNO)
Define_Map For(*INPUT) Class(#EMPNO) Name(#EMPLOYEE)
Define_Com Class(#empno) Name(#wempno)
#std_num := #EMPLOYEE.value.CurSize
#wempno := #EMPLOYEE.value
```

for each(#entry) in(#Collections.Employees)
#wempno := #entry.value.Substring( 1 #std_num )

#empno := #entry.value
#fullname := #Collections.Fullnames<#empno>
Add_Entry To_List(#LISTVIEW)
#LISTVIEW.Currentitem.Image <= #VI_EMPLOY</pre>

```
* select current employee
If ('#employee = #wempno')
set #listview.CurrentItem selected(true)
Endif
endfor
```

Endroutine

Ptyroutine Name(get_empno) Define_Map For(*output) Class(#EMPNO) Name(#EMPLOYEE) #EMPLOYEE := #empno Endroutine

* Close Form and return result Evtroutine Handling(#PB_OK.Click #LISTVIEW.DoubleClick) #COM_OWNER.hideform #COM_OWNER.Modalresult := OK Endroutine End_Com

- 3. Save the form.
- 4. Create a new Reusable Part / Object.

This stores a keyed collection of employee numbers and a keyed collection of full names, keyed on EMPNO.

This design enables the prompt form to load the collections once, the first time it is used. Subsequent executions of the prompt form do not need to read the employee file.

Name: iiiVFW27

#### Description: Employees Collection

5. Complete the reusable part's initial code by copy and pasting the following code:

* Keyed collections DEFINE_COM CLASS(#PRIM_KCOL<#EMPNO #Empno>) NAME(#Empl-DEFINE_COM CLASS(#PRIM_KCOL<#FULLNAME #EMPNO>) NAME(#

* Properties DEFINE_PTY Employees GET(*COLLECTION #Employees) DEFINE_PTY Fullnames GET(*COLLECTION #Fullnames)

* Load keyed collection with Personnel details EVTROUTINE HANDLING(#COM_OWNER.CreateInstance) OPTIONS(*N Select Fields(#empno #givename #surname) From_File(PSLMST) #Employees<#Empno> := #Empno #Fullnames<#Empno> := #Givename + ' ' + #Surname Endselect ENDROUTINE

- 6. Compile the reusable part.
- 7. In the editor, switch to the form, iiiVFW26 and change this line, to match your reusable part name:

DEFINE_COM CLASS(#iiiVFW26) NAME(#Collections) scope(*Application

8. Compile the form iiiVFW26.

**Note:** The form contains the following event routine:

* Close Form and return result Evtroutine Handling(#PB_OK.Click #LISTVIEW.DoubleClick) #COM_OWNER.hideform #COM_OWNER.Modalresult := OK Endroutine

The routine executes when the *OK* button is clicked or a list entry is double clicked.

The *Com_Owner.ModalResult* := *OK* informs the calling field that the prompter is being closed successfully and it should retrieve the value of the default property containing the selected employee number.

- 9. Open the field iiiEMPNO in the editor. Select the Visualization tab and click the *Insert Prompter Form* toolbar button. In the *Repository Find* dialog, select your prompter form iiiVFW26 and click *OK*.
- 10. Select the *VisualEdit* component and use the *Details* tab to change its *ShowPrompter* property to **True**. Note that the *VisualEdit* visualization now shows a prompter button.
- 11. Save and close the field definition.
- 12. Open the *Resources for a Section* reusable part (iiiVFW25) in the editor and select the *Design* view.
- 13. Select the Code column (iiiEMPNO). The column should already have a *ReadOnly* property of **False**.

Make the following changes:

Property	Value
EditorPart	iiiVFW25
DisplayAppearance	Edit
EditAppearance	Edit

- 14. Re-compile the reusable part.
- 15. Execute the Framework and select the Resources command handler for a

section.

When you click in the employee number column (Code) a prompt button will now be shown.

- 16. Click the prompt button to run the prompt form. Note that the employee number for this row is selected in the list.
- 17. Select a new employee number and click *OK* or double click the entry. The form closes and the column entry is updated.

This reusable part prompter visualization can also be used on a form or panel.

## Summary

### **Important Observations**

- Field Visualization provides a simple consistent interface for activities such as prompting. However, for very complex prompting requirements, you will most likely need to create another component.
- Using reusable parts and prompter forms ensures that the same field provides a consistent interface wherever used.
- Having multiple visualizations for the same field allows you to turn off/change the style of field for particular situations.

# **Tips & Techniques**

• When creating a prompter form you must ensure that you follow the prompter form guidelines. When a a property is added to the form, ensure it is made the default property for the form, such as DefaultPty(p_Employee_Number) in this example:

DEFINE_PTY NAME(P_EMPLOYEE_NUMBER) GET(*AUTO #EMPNO)

• The **SET** of this property will be called when the user activates the prompter (via either F4 or the ellipses button).

# What I Should Know

- How to define list column properties to provide check box, combo box, image and text and multiline edit visualization.
- Input capable Date and DateTime columns will show a date prompt by default.
- How to define a prompter field visualization.
- How to implement a prompter form visualization in a list column.

# VFW104 – Simple Keyed Collections

## **Introduction to Collections**

Collections are conceptually similar to conventional arrays. In Visual LANSA, they constitute a very powerful way of grouping components of the same type. You can collect any Class of component in a Collection. Commonly used types of collections in Visual LANSA include:

- Keyed Collection
- List Collection
- Array Collection
- Sorted Array Collection.

In this exercise, you will use a keyed collection.

Elements in conventional arrays are accessed using the element's index position, usually a number. In Visual LANSA, an element in a Keyed Collection can be accessed through its numeric index position or using an alphanumeric string.

A keyed collection is defined using the following command:

#### DEFINE_COM CLASS(#PRIM_KCOL<<Collecting><Key Value>>) NAME(...) STYLE(...)

The important properties in a Keyed Collection are:

- *Class:* The value #PRIM_KCOL defines the component as keyed collection.
- *Collecting:* The class of component or class of each element in the group.
- *Key Value:* The numeric or alphanumeric value that allows you to access one element in the collection.
- *Style:* Factory (default) means a new instance of the type of component should be created whenever the collection is accessed. A *Style:* Collection means a *Null reference is to be returned whenever the collection is accessed with a key that does not exist.

The Style of collection you choose to use depends on whether the information stored in the collection is going to be shared by other forms or components and also on what type of thing you are going to collect. Sometimes a working list or an array would provide the same functionality as a collection, but it is impossible to pass a working list or an array to another form or reusable component. Storing the data in a collection would solve this issue. If you want to collect bitmaps or radio buttons then a Keyed Collection is the only option. If you are collecting objects of any kind you should use a Style Collection. For example, when you define a component in an event routine and then place a reference to it in a collection Style Collection, when the routine terminates the object ceases to exist. Or, you can explicitly destroy the object by setting its reference to *NULL. You cannot do that to an object created in a Style Factory collection. The object would exist in memory until the application is closed. List collections are defined by using the following command:

#### DEFINE_COM CLASS(#PRIM_LCOL<< Collecting >>) NAME(...)

The important properties in a list collection are:

- *Class:* The value #PRIM_LCOL defines the component as a list collection.
- *Collecting:* The class of component or class of each element in the group.

# Objective

- To learn about using keyed collections.
- To develop an *Images* command handler for Sections, which displays images for the employees in the section.
- To use the VL Framework *avSwitch* method to link an employee image component to the Employee business object *Details* command handler.

🖻 🗆 • 🤱 🔟 😳 • 🗖	ē -			
Department	Description	Address Line 1	Address Line 2	
ADM	ADMINISTRATOR	125 Main St,	Blacktown	1
E Sections				J.
INTERNAL ADMIN SRV	01	125 Main St,	Blacktown	1
PURCHASING SECTION	02	123 Pacific Highway,	North Sydney, 2000	1
ACCOUNTING SECTION	03	252 Canterbury R	CANTERBURY.	I.
PERSONNEL SECTIONXX	06	121 Smith St	Newtown	U
VEHICLE MAINTENANCE	09	121 Smith Street	Newtown	
AUD	INTERNAL AUDITING	125 Main St,	Blacktown	
FLT	FLEET ADMINISTR	125 Main St,	Blacktown	
GAC	GROUP ACCOUNTS	125 Main St,	Blacktown	-
<			3	aŤ
💭 Section : Images (INTERNAL	ADMIN SRV-01)			F
Detais S Employees II In	nages 🔲 Picture 🦯 🔁 Tran	sfer		
Smithe Smithe Dougl	as Mrs brick Hedto	rd		
		Employee Detail	s	

To achieve these objectives you will complete the following steps: Step 1. Create the Employee Images for Section Command Handler Step 2. Create an Employee Image Component

Step 3. Make Sections Business Object Visible

Step 4. Implement the Employee Image component.

Step 5. Record the Switch History using the Virtual Clipboard

Step 6. Use the Switch History to Return to the Original BO Summary

# **Before You Begin**

This exercise depends on VFW080 – Using an Explorer Component, which maintains employee image records in file iiiEmpImages.

# **Step 1. Create the Employee Images for Section Command Handler**

This component will dynamically create a keyed collection of image components (PRIM_IMGE) keyed by EMPNO for the employees in a department / section. Employee images are retrieved from the file iiiEmpImages.

The images are displayed on the command handler panel, managed by a *Flow Across* layout manager.

The component will also create a keyed collection of flow item managers (PRIM_FWLI) keyed by EMPNO which position each image across the panel.

1. Create a new *Reusable Part / Panel*:

#### Name: iiiVFW28

#### Description: Employee Images for a Section

- 2. Give the RP an ancestor of VF_AC010.
- 3. Use the *Design* ribbon to add a *Flow Across* manager to iiiVFW28.
- 4. Switch to the *Source* tab and define a keyed collection, to collect PRIM_IMGE keyed by EMPNO.

Define_Com Class(#PRIM_KCOL<#PRIM_IMGE #empno>) Name(#ImageC

The *Reference*(**dynamic*) means that the collection will exist only once you use *SET_REF* to create it.

5. Similarly define a keyed collection, to collect PRIM_FWLI keyed by EMPNO.

Define_Com Class(#PRIM_KCOL<#PRIM_FWLI #EMPNO>) Name(#Image

6. Create a *uExecute* method routine which redefines the ancestor's method, and invoke the ancestor *uExecute* method.

Retrieve the current instance list entry using the List Manager component and retrieve Akey1 and Akey2.

The *uExecute* method is called when an entry in the instant list is selected.

Mthroutine Name(uExecute) Options(*redefine) #com_ancestor.uExecute
#avlistmanager.getCurrentInstance Akey1(#deptment) Akey2(#section) Endroutine

7. Add the following code to the *uExecute* routine:

* clear existing collections Set_Ref Com(#ImageCollection) To(*null) Set_Ref Com(#ImageFlowCollection) To(*null) * Create collections dynamically Set_Ref Com(#ImageCollection) To(*Create_as #PRIM_KCOL<#prim_IMGE Set_Ref Com(#ImageFlowCollection) To(*Create_as #PRIM_KCOL<#prim_I

This clears existing collections (from a previous execution).

The *ImageCollection* and the ImageFlowCollection are then created dynamically using SET_REF.

8. Add the following code to populate the keyed collections:

* Load images into panel Select Fields(#empno) From_File(pslmst1) With_Key(#deptment #section) Fetch Fields(#iiiempimg) From_File(iiiEmpImages) With_Key(#empno) Val_I If_Status Is(*okay) If (*Not #iiiempimg.filename.isnull) If_Ref Com(#ImageCollection<#EMPNO>) Is(*NULL) Set_Ref Com(#ImageCollection<#EMPNO>) To(*CREATE_AS #PRIM_IMG Set_Ref Com(#ImageFlowCollection<#EMPNO>) To(*CREATE_AS #PRIM_ Set Com(#ImageCollection<#EMPNO>) Parent(#COM_OWNER) Filename(# Set Com(#ImageFlowCollection<#EMPNO>) Parent(#FWLM_1) Manage(#In Endif Endif Endif Endif Endif Endif

9. Review this new logic:

Employee numbers are retrieved from the logical file PSLMST1 using the keys retrieved from the instance list.

An employee image is read from the file iiiEmpImages.

Field iiiEMPIMG is a BLOB field. Retrieving it from a file, restores the

image file to a local folder. The path and file name are held in *iiiEMPIMG.filename*.

If this employee has an entry in iiiEmpImages, the *IF_REF* checks an entry for this employee doesn't exist in the *ImageCollection*.

The *SET_REF* then creates an entry of PRIM_IMGE in *ImageCollection*, keyed on EMPNO.

Another *SET_REF* creates an entry for PRIM_FWLI in *ImageFlowCollection*.

The *SET Com(ImageCollection* ) . . . sets the Parent and Filename property for this collection entry.

The Parent must be #COM_OWNER for the image to be shown on the RP's panel.

Similarly the next *SET*, sets the *ImageFlowCollection* entry Parent to FWLM_1, the flow across manager and the Manage property to this *ImageCollection* entry.

The *#COM_OWNER.Realize* makes the panel components visible once they have all been created.

- 10. Compile your new command handler, iiiVFW28 *Employee Images for a Section*.
- 11. Start the VL Framework as Designer.
- 12. Open the *Properties* dialog for _Sections by double clicking on a section entry in the instance list.
- 13. Select the *Commands Enabled* tab, select the *Images* action and plug in iiiVFW28.
- 14. Save and Restart the VL Framework.
- 15. Use the *By Location* filter for employees and then use the *Images* command handler to ensure that images have been saved for a number of employees in this department / section (for example, department ADM and section 01).

Remember to select the Clear List checkbox before rebuilding the list of employees.

16. Now select the Departments business object and expand ADM to display the first section (Internal Admin SRV) and select this section. Select the *Images* command handler. Your *Images* command handler should look like the

# following:

🖻 🗆 • 🤱 🔟 😂 • 🖬	e -				
Department	Description	Address Line 1	Address Line 2	Address Line 3	
ADM     Sections	ADMINISTRATOR DEPT				m
INTERNAL ADMIN SRV	01	125 Main St,	Blacktown	NSW	
PURCHASING SECTION	02	123 Pacific Highway,	North Sydney, 2000	NSW	
ACCOUNTING SECTION	03	252 Canterbury Road,	CANTERBURY.	NSW.	-
4	III				P
					_
					X
Section : Images (INTERNAL	ADMIN SRV-UTJ				
🗖 Details 🛛 🔗 Employees 🚺 Im	iages 🛛 📴 Picture 🛛 🦰 Trans	sfer			
	<b>.</b>				

## Step 2. Create an Employee Image Component

In the following steps you will enhance the functionality of the Employee Images for Section command handler. You will do this as follows:

- Create an Employee Image reusable part which displays the employee image in a Group Box, with a Caption showing the employee surname
- The *Employee Images for Section* command handler will be changed to create a collection of the new *Employee Image* components.
- In a later step, the *Employee Images for Section* command handler will be enhanced to use the Framework's switch service when an image is double clicked or when a pop-up menu is used, to display the *Employee Details* command handler for the selected employee.
- 1. Create a new Reusable Part / Panel:

#### Name: iiiVFW29

### Description: Employee Image

- 2. On the *Design* tab, resize the panel as shown.
  - a. Drag and drop a *Group Box* onto the panel.
  - b. Drag and drop an *Image* component onto the *Group Box*.

Your design should now look like the following:

Design Sour	ce	М
IIIVFW3		
•		

- 3. Drop a Pop-Up Menu component onto the image and define one menu item as **Show Employee Details**
- 4. Save the new reusable part.
- 5. Switch to the *Source* tab, and define the following properties:

Define_Pty Name(uCaption) Set(SetCaption) Define_Pty Name(uEmpNum) Get(*auto #empno) Set(*auto #empno) Define_Pty Name(uFileName) Set(SetFileName) Ignore the errors for the property routines which you will shortly create.

The *Employee Images for Section* command handler needs to set these properties to set up the image to be displayed.

6. Add the following event definition, which passes employee number:

Define_Evt Name(uShowEmpDetails)
Define_Map For(*input) Class(#empno) Name(#uEmpNum)

The event will be signaled when an image is double clicked, or the pop-up menu item is clicked.

7. Complete the Employee Image component definition with the following code:

* Set Group Box Caption to Surname Ptyroutine Name(SetCaption) Define_Map For(*input) Class(#surname) Name(#i_Caption) #GPBX_1.caption := #i_Caption Endroutine * Set image path/filename Ptyroutine Name(SetFileName) Define_Map For(*input) Class(#std_strng) Name(#i_filename) #IMGE_1.fileName := #i_filename Endroutine * Signal uShowEmpDetails Evtroutine Handling(#IMGE_1.DoubleClick #MITM_1.Click) Options(*NOC Signal Event(uShowEmpDetails) Uempnum(#empno) Endroutine

- 8. Review the code which you just added.
  - a. The *SetCaption* property routine sets the Group Box caption. The property is passed employee surname.
  - b. The *SetFileName* property routine sets the Image filename property. The property uFileName is passed the image file name.
  - c. An event routine for *Image DoubleClick* and *Menu Item 1 Click* signals the uShowEmpDetails event and passes employee number. The *uEmpNum* property automatically sets and gets the value of employee number (EMPNO).

9. Compile the Employee Image component.

## Step 3. Make Sections Business Object Visible

In the following steps you will change the Employee Images for Section command handler to be able to switch to the Employee Details command handler. The Employee Details command handler will be enhanced to enable the user to return to the Employee Images for Section command handler.

**Note:** Switching can only be performed on objects that are visible in the Navigation panel.

- 1. In the VL Framework, display the properties of the *_Departments* business object.
- 2. Display the Instance List / Relationships tab.
- 3. Select *Sections* (*_SECTION*) in the business objects list at the bottom left of the panel.
- 4. Select the option _Sections Allow Selection from Navigation Pane.



- 5. Close the properties of the *_Departments* business object. The *_Sections* business object is now visible in the navigation pane.
- 6. Display the properties of the *_Sections* business object.
- 7. Select the *Icons* tab, and select any suitable icon for _*Sections*.

Next you need to replace the mock-up filter in the *_Sections* business object with a functional filter to populate the instance list:

- 8. Start the *Program Coding Assistant*.
- 9. Select the *_Sections* business object in the iii HR application.
- 10. Select New Filter / Windows as the platform and a Filter that searches a file

or a view.

- 11. Click *Next*.
- 12. Specify SECTAB as the physical file, and DEPTMENT and SECDESC as the *Visual Identifiers*.

Spe res bas he p ese	ecify the identific embles this busin ic identification p physical file that mbles this busine UALIDENTIFIER:	ation protocol ness object sp protocol for yo most closely iss object is: S (for building	you have decided to use for th ecify its name and the assistant ou. SECTAB Section VisualID1 and VisualID2 values)	is business will attem	object. If a physical file pt to automatically deduce a le	
	Field Name	Туре	Description		Drop Selected	
1	DEPTMENT	ALPHA	Department Code			
2	SECDESC	ALPHA	Section Full Description	E	Drop All	
3						
4				-		
5				-		
RC	fields from this P DGRAMMATIC ID Field Name	hysical File ENTIFIERS (fo Type	Add Keys or building AKey1,2,3,4,5 and N Description	Key1,2,3,4	dd All	
1	DEPTMENT	ALPHA	Department Code	-		
2	SECTION	ALPHA	Section Code	1	Drop All	
3				-		
_						

- 13. Accept the other defaults set by the *Program Coding Assistant* and click *Next*.
- 14. Specify DEPTMENT field as the key to be used for search operations.
- 15. Select Allow Generic searching, Remember key values between filter executions and Allow user to clear instance list.
- 16. Click Next.
- 17. Select Routine to listen for signals to update the instance list.
- 18. Click Generate Code.
- 19. On the *Generated Code* page specify **iiiVFW30** as the name of your filter and **Sections Filter** as the description. (**iii** are your initials. If you are using an unlicensed or trial version of Visual LANSA, your component names must have the form DEMCOM01 to 09).
- 20. Click *Create*. The component is displayed in the Visual LANSA Editor.
- 21. Compile the filter.
- 22. In the Framework, snap the filter in the Sections business object. Use the

*Find* dialog and select your reusable part to snap in. This will snap in using the *Identifier*.

23. *Save and Restart* the VL Framework and test the filter. You will now be able to access the *Sections* business object and command handlers directly from the Navigation pane.

## Step 4. Implement the Employee Image component.

In this step you will change the *Employee Images for Section* command handler (iiiVFW27) to use the new Employee Image component.

- 1. Open iiiVFW28 in the editor.
- 2. Change the *ImageCollection* definition to use #iiiVFW29 (Employee Image). Changes are highlighted in red, ignore errors at this stage.

Define_Com Class(#PRIM_KCOL<#iiivfw28 #empno>) Name(#ImageCollec

3. Change the *uExecute* method routine:

Change the *Set_Ref* which adds an image entry to *ImageCollection*, to add the image component, *iiiVFW28*.

Set_Ref Com(#ImageCollection<#empno>) To(*create_as #iiivfw28)

4. Change the *Set*, which sets the properties for the current *ImageCollection* entry, to set the employee number, filename and caption properties for the new image component:

Set Com(#ImageCollection<#EMPNO>) Parent(#COM_OWNER) uFilename

5. Add the following method to convert to proper case:

Mthroutine Name(uProperCase) Define_Map For(*result) Class(#std_name) Name(#o_text) Define_Map For(*input) Class(#surname) Name(#i_text) #std_name := #i_text.lowerCase #std_flag := #std_name.substring( 1, 1 ).upperCase #o_text := #std_flag + #std_name.substring( 2, 19 ).trim Endroutine

6. Retrieve Surname from the employee file and convert to proper case:

Select Fields(#empno #surname) From_File(pslmst1) With_Key(#deptment #:
#std_name := #com_owner.uProperCase( #surname )

7. Add logic to switch to the Employee business object, *Details* command handler when an image component signals *uShowEmpDetails* event:

Evtroutine Handling(#ImageCollection<>.uShowEmpDetails) Uempnum(#I_E #empno := #I_EmpNum * Switch to Employee / Details command handler #avframeworkmanager.avSwitch To(BUSINESSOBJECT) Named(EMPLOYE Endroutine

**Note:** The *To(), Named()* and *Execute()* parameters are case sensitive. Ensure that they match your Framework object names.

8. Add a VL Framework event handling routine for *avAddSwitchInstances* which will tell the *Employees* business object which instance should be displayed based on the employee number (*uEmpNum*) passed by the *uImageClicked* event.

**Note:** This routine sets up the required instance list columns for Employee to invoke the *avAddSwitchInstance*.

Evtroutine Handling(#avFrameworkManager.avAddSwitchInstances) Options( Define Field(#udate) Type(*char) Length(19) * Make sure the caller is this component If_Ref Com(#Caller) Is_Not(*Equal_to #Com_Owner) Return Endif Fetch Fields(#surname #givename #deptment #salary #startdte) From_File(psli #fullname := #surname + ', ' + #givename #udate := #startdte.asdate( DDMMYY ).AsDisplayString( DDsMMsCCYY ) Invoke Method(#avFrameworkManager.avAddSwitchInstance) Businessobject Endroutine

The *avAddSwitchInstances* event routine is always executed immediately after you execute a switch using the *avSwitch* method. This event allows you to control what data will be placed in the instance list of the target business object. The component signalling this event is passed in the Caller parameter.

It is important to only execute the code in this event if the component that signalled this event is the component itself. Therefore you should return from this event routine if the caller is not equal to #com_owner. Notice how the *is_not(*Equal_to)* is used to compare the #*Caller* and #*Com_Owner*. You must use this syntax due to the fact that you are comparing the component itself and not a simple string.

The *avAddSwitchInstance* method specifies what data to add in the target

instance list.

If required, you could call the *avAddSwitchInstance* method repeatedly, to place multiple entries into the target business object's instance list.

- 9. Compile the enhanced *Employee Images for Section* command handler (iiiVFW28).
- 10. Execute the Framework. Select the *Images* command handler for the Department and Section for which you created employee image records. Your *Images* command handler should now look like the following:



11. Double click on an employee image or right click and use the pop-up menu to *Show Employee Details*.

The Framework should switch to the *Details* command handler for the selected employee (check the surname shown).

The instance list should contain an entry for this employee and the details of the employee should be displayed:

	x			x
By Name 100 By Locatio	n		۶ 🗂	
🗸 Clear List	Search	Employee Full Name	Depar	Salary Start Da
Employee Surname		A1020 DOUGLAS, ADAM PETER	ADM	121,500
SM				
				•
				x
Employee : Details (A1)	020-DOUGLAS, ADAM PETER)			
Details 😥 Brief Notes	🚺 Images 🛛 📎 Notes 🛛 🛗 Skil	ls 2 🦻 Skills		
Employee Number	A1020	Employee Salary	121,500.04	Save
Employee Surname	DOUGLAS	Start Date (DDMMYY)	1/02/88	
Employee Given Name(s)	ADAM PETER	Termination Date (DDMMYY)	0/00/00	
Street No and Name	6 Reading Avenue,			
Suburb or Town	KINGSLANGLEY.			
State and Country	NSW.			
Post / Zip Code	2147			
Home Phone Number	674 5310			
Business Phone Number	639 5188			
Departments	ADMINISTRATOR DEPT	w		
Sections	INTERNAL ADMIN SRV	×		

## Step 5. Record the Switch History using the Virtual Clipboard

In this step you will record the switch history using the VL Framework's virtual clipboard so that the end-user will be able return to the object that initiated the switch.

To use the virtual clipboard most effectively, you need to devise a standardized naming protocol for items that are posted onto it. In this exercise you will use this standard to store the switch history:

ID1	SWITCH_HISTORY
ID2	Target Business Object Name
ID3	Target Command Name
ID4	OBJECT_NAME or COMMAND_NAME
Γ Δ. 7. ]	(abiant an annual name)

FromAValue <object or command name>

In effect you will be storing a switch history table on the Framework's clipboard. The first key or ID is the code 'SWITCH_HISTORY' to indicate that all records with this ID are related to switching history.

The ID2 and ID3 contain the business object and command respectively that you are switching to.

ID4 contains where you came from. Therefore you need to add two records to the virtual clipboard:

- one where ID4 equals OBJECT_NAME (the business object)
- and another where ID4 equals COMMAND_NAME (the command).
- 1. Display the *Source* tab for the *Employee Images for Section* command handler (iiiVFW28).
- 2. In the event routine for *#ImageCollection*<>.*uShowEmpDetails* add code to add the appropriate records to the switch history. Changes are highlighted in red.

Evtroutine Handling(#ImageCollection<>.uuShowEmpDetails) Uempnum(#I_ #empno := #I_EMPNUM

* Save to clipboard return list

* Returns to Section / Images for parent department #avframeworkmanager.avsavevalue Withid1(SWITCH_HISTORY) Withi #avframeworkmanager.avsavevalue Withid1(SWITCH_HISTORY) Withi * Switch to Employee / Details command handler

#avframeworkmanager.avSwitch To(BUSINESSOBJECT) Named(EMPLOYE Endroutine

**Note:** Ensure that *Withid2() Withid3()* and *FromValue()*, all use your business object names and command name.

3. Compile the command handler iiiVFW28.

## Step 6. Use the Switch History to Return to the Original BO

In this step you will change the Employee Details command handler to use the switch history to allow the end-user to return to the Departments business object from where they initiated the switch.

- 1. Open the Employees' *Details* command handler iiiVFW06.
- 2. Drag a push button below the *Save* button.
- 3. Make the *Caption* of the button **Back to Sections**.
- 4. Make the *Name* of the button, **BACK_BTN**.
- 5. Create a *Click* event for the button.
- 6. In the *BACK_BTN.Click* event add this code so that when the users click on the button, they will be switched back to the business object from which they came:

EVTROUTINE HANDLING(#BACK_BTN.Click)

define field(#ff_objnme) TYPE(*CHAR) LENGTH(32) DESC('Object Name') define field(#ff_cmdnme) TYPE(*CHAR) LENGTH(32) DESC('Command Na * Determine the business object name to switch to

#avFrameworkManager.avrestorevalue WithID1(SWITCH_HISTORY) WithII
* Determine which command within the business object to switch to

#avFrameworkManager.avrestorevalue WithID1(SWITCH_HISTORY) WithII
* Perform the switch

* Retruns to Departments / Details

#avframeworkmanager.avSwitch To(BUSINESSOBJECT) NAMED(#ff_objnr ENDROUTINE

To send the user back to the component from which the switch occurred:

- You need to look at the switch history on the virtual clipboard. Remember that you need to retrieve both the business object and the command to which you need to return. That requires retrieving two values from the virtual clipboard.
- The code first retrieves the OBJECT_NAME or business object value and then the COMMAND_NAME or command value.
- Remember that you don't want to hard code the component name, which is why *avObjectType* (business object name) and *avCommandType* (command name) were used as the values to the Withid2() and *Withid3()*

parameters.

• When you have these two values you can perform another switch to return to the previous component.

In the preceding code, the business object was retrieved into the *#ff_objname* field and the command was retrieved into the *#ff_cmdnme* field. Now you simply use the same technique learned earlier to switch to a business object and execute the command.

Your code should look like this:

```
Define Handling(#BACK_BTN.Click)
Define Field(#ff_objnme) Type(*CHAR) Length(32) Desc('Object Name')
Define Field(#ff_cmdnme) Type(*CHAR) Length(32) Desc('Command Name')
* Determine the business object name to switch to
#avFrameworkManager.avrestorevalue Withid1(SWITCH_HISTORY) Withid2(#com_owner.Avobjecttype)
Withid3(#com_owner.Avcommandtype) Withid4(OBJECT_NAME) Toavalue(#ff_objnme)
* Determine which command within the business object to switch to
#avFrameworkManager.avrestorevalue Withid1(SWITCH_HISTORY) Withid2(#com_owner.Avobjecttype)
Withid3(#com_owner.Avcommandtype) Withid4(COMMAND_NAME) Toavalue(#ff_cmdnme)
* Perform the switch
* Saved values set up to return to parent level - department
#avframeworkmanager.avSwitch To(BUSINESSOBJECT) Named(#ff_objnme) Execute(#ff_cmdnme) Caller(#com_owner)
Endroutine
```

7. Recompile the iiiVFW06 command handler.

You are now ready to test using switch history:

- 8. In the VL Framework, select _Sections and populate the instance list and select a Section with employee images recorded.
- 9. Double click an employee image in the Employee Images for Section command handler, to display details of the selected employee.
- 10. On the *Details* command handler of the Employees business object, click on the *Back to Sections* button to return to the _Sections business object.



## Summary

## **Important Observations**

- The VL Framework switching service allows your filters and command handlers to switch control between different business objects and to execute commands at the Framework, application or business object level.
- Switching mimics the actions that a user would perform. The target business object must be able to be selected from the menu. That is, at the time the switch occurs, the option Allow selection from the navigation pane in the target business object properties should be checked (selected), and the user should be authorized to the business object.
- You can use the Virtual Clipboard for remembering and exchanging information.
- To use the virtual clipboard most effectively you need to devise a standardized naming protocol for items that are posted onto it.

## **Tips and Techniques**

• In the shipped examples, the *Advanced* section of the *Programming Techniques* sample application demonstrates switching and remembering values (virtual clipboard).

## What I Should Know

- How to switch between business objects.
- How to use the virtual clipboard to record switch history so that the endusers can switch back to object where the switch was initiated.

## VFW106 – Using a List Collection

## Objective

To build a *Monthly* command handler for the *Reports* business object which displays a list of employees. This doesn't represent a real application. The Monthly command handler is simply a mechanism to demonstrate passing multiple references using a list collection. A reference to the list collection will be passed to a simple form which will show the current entries in the list collection.

💽 Report : Monthly			
📃 Monthly 📄 Weekly			
Employee Number	Employee Su	rname	Employee
A0070	BROWN		VERONIC
A0090	BLOGGS	Show Employee List Collection	RED JO
A0193	SIMPSON		RED
A0907	JONES	Employee full name	INNE
A1001	JONES	JOHN SMYTHE	EN
A1002	SMYTHE	PETER SMITHS	OHN
A1003	SMITHE	GEORGE SNELL	tobert
A1004	SMITHSON	WILLIAM PERRY	AUL
A1005	SMITHS	CHRISTOPHER PERRIN	ETER
A 1006	SMITHERS		ACK
A1007	SNELL		EORGE
A 1008	SNEDDON		ILLAN
A 1009	SNASHALL		PAMIAN
A 10 10	PERRY		VILLIAM
A1011	PERRIN		CHRISTO
A1012	PAUL		ATRICK
A1013	PATTISON		EORGE
A1014	MOORE		OHN

To achieve this objective you will complete the following steps:

- Step 1. Create the Employee Object.
- Step 2. Create the Monthly Command Handler
- Step 3. Create the Selected Employees Viewer
- Step 4. Complete Monthly Command Handler
- Summary

## **Before you Begin**

You must complete exercises VFW030, VFW040, VFW042 and VFW104.

## **Step 1. Create the Employee Object.**

The Employee Object is a simple non visual component that stores values for each employee.

1. Create a new *Reusable Part / Object*:

Name: iiiVFW31

#### Description: Employee Object

2. Define the properties this object needs to store the employee data, allowing values to be set and retrieved (get).

Define_Pty Name(UEMPLOYEENUMBER) Get(*AUTO #EMPNO) Set(*AU Define_Pty Name(UEMPLOYEESURNAME) Get(*AUTO #SURNAME) Set Define_Pty Name(UEMPLOYEEGIVENAME) Get(*AUTO #GIVENAME) Set(*AU Define_Pty Name(UEMPLOYEESALARY) Get(*AUTO #SALARY) Set(*AU

3. Compile the reusable part.

## Step 2. Create the Monthly Command Handler

1. Create a new Reusable Part / Panel:

Name: iiiVFW32

### Description: Monthly Command Handler for Reports

- 2. Give the reusable part an ancestor of VF_AC010
- 3. Use the *Design* ribbon to give iiiVFW32 an *Attachment manager*.
- 4. Add a List View to the center of the Panel. Change the List View *Name* property to **EMP_VIEW**.
- 5. Create an *Initialize* event for EMP_VIEW.
- 6. Locate the file PSLMST on the *Repository* tab and drag the fields EMPNO, SURNAME, GIVENAME and SALARY onto the list.
- 7. Open the *Source* tab and define a key collection of employee objects, iiiVFW31 keyed by EMPNO

Define_Com Class(#PRIM_KCOL<#iiivfw31 #EMPNO>) Name(#EMPLOYI

8. Add logic to the *Initialize* event for EMP_VIEW to populate the list view with all records from file PSLMST.

Clr_List Named(#EMP_VIEW) Select Fields(#EMP_VIEW) From_File(pslmst) Add_Entry To_List(#EMP_VIEW) Endselect

9. Add a *Set_Ref* to create an entry of iiiVFW31 to the keyed collection, for each employee, keyed by EMPNO, setting the properties of the employee object for each employee.

The new code is highlighted in red.

Clr_List Named(#EMP_VIEW) Select Fields(#EMP_VIEW) From_File(pslmst) Add_Entry To_List(#EMP_VIEW) Set_Ref Com(#EMPLOYEES<#EMPNO>) To(*CREATE_AS #IIIVFW31 #EMPLOYEES<#EMPNO>.UEMPLOYEENUMBER := #EMPNO #EMPLOYEES<#EMPNO>.UEMPLOYEEGIVENAME := #GIVENAMI #EMPLOYEES<#EMPNO>.UEMPLOYEESURNAME := #SURNAME

#### **#EMPLOYEES<#EMPNO>.UEMPLOYEESALARY := #SALARY** Endselect

10. Create a method routine, *SelectionChanged* which will be called each time a list view entry is selected or loses selection.

The routine should:

- a. Define a list collection of the employee object iiiVFW31. This will be used to store a list of the currently selected employees.
- b. Read through the list view using Selectlist
- c. Go straight to read the next list view entry, if it is not currently selected, using the Continue statement.
- d. Insert an entry to the list collection, from the keyed collection EMPLOYEES

Your code should look like the following:

```
Mthroutine Name(SelectionChanged)
Define_Com Class(#PRIM_LCOL<#iiivfw31>) Name(#SELECTION)
Selectlist Named(#EMP_VIEW)
Continue If(*Not #EMP_VIEW.CURRENTITEM.SELECTED)
#SELECTION.INSERT( #EMPLOYEES<#EMPNO> )
Endselect
* Show the selected employees form
Endroutine
```

Note the comment line. You will create the selected employees form in the next step.

The list collection is defined within the method routine and is destroyed when the routine ends.

11. In the Design view, select the list view and create event routines for ItemGotSelection and ItemLostSelection. Add code to each routine to invoke the SelectionChanged method.

Your code should look like the following:

Evtroutine Handling(#EMP_VIEW.ItemGotSelection) Options(*NOCLEARM #com_owner.SelectionChanged Endroutine

Evtroutine Handling(#EMP_VIEW.ItemLostSelection) Options(*NOCLEARM #com_owner.SelectionChanged Endroutine

12. Leave this reusable part open in the editor.

## Step 3. Create the Selected Employees Viewer

1. Create new *Form / Basic Form*:

Name: iiiVFW33

Description: Selected Employees Viewer

- 2. Size the form approximately to *Height* **430** and *Width* **230**.
- 3. Change the form's *FrameStyle* to **Dialog**, and *FormStyle* to **StayOnTopChild**
- 4. Use the *Design* ribbon to give the form an *Attachment manager*.
- 5. Drop a *List View* into the center of the form and change the *List View's Name* to **SEL_LIST**.
- 6. Drag and drop the field Fullname onto the list view, and change its *WidthType* to **Remainder**.

Your form should look like the following:



- 7. Save the form
- 8. Create a method routine named *uShow*.

This method will be invoked by the *SelectionChanged* method in the Monthly Command Handler for Reports - iiiVFW32.

The *uShow* method needs to do the following:

- a. Define an input map of Class(#PRIM_LCOL<iiiVFW31>), which is passed by reference.
- b. Clear the selected employees list view, SEL_LIST
- c. Read through the list collection EMPLOYEES using a For/Endfor loop.
- d. For each entry set Fullname from the uEmployeeGivename and uEmployeeSurname properties
- e. Add an entry to SEL_LIST
- f. After processing the list collection show this form.

Your code should look like the following:

Mthroutine Name(uShow) Define_Map For(*INPUT) Class(#PRIM_LCOL<#iiivfw31>) Name(#EMPLC Clr_List Named(#SEL_LIST) For Each(#EMPLOYEE) In(#EMPLOYEES) #FULLNAME := #EMPLOYEE.UEMPLOYEEGIVENAME + ' ' + #EMPLO` Add_Entry To_List(#SEL_LIST) Endfor #COM_OWNER.SHOWFORM Endroutine

9. Compile the form iiiVFW33.

## **Step 4. Complete Monthly Command Handler**

- 1. Switch to the reusable part iiiVFW32 in the editor (*Monthly Command Handler for Reports*).
- 2. In the *Design* view, drag and drop the *Selected Employees* form, iiiVFW33 onto the panel. This will create a *Define_Com* for the component. Change its *Name* to **#Selected_Employees**.
- 3. Complete the *SelectionChanged* method to invoke the *Selected_Employees* form's *uShow* method, passing the list collection (with the name SELECTION).

Your code should look like the following. New code is highlighted in red.

Mthroutine Name(SelectionChanged) Define_Com Class(#PRIM_LCOL<#iiivfw33>) Name(#SELECTION) Selectlist Named(#EMP_VIEW) Continue If(*Not #EMP_VIEW.CURRENTITEM.SELECTED) #SELECTION.INSERT( #EMPLOYEES<#EMPNO> ) Endselect * Show the selected employees form **#Selected_Employees.uShow( #SELECTION )** Endroutine

- 4. Compile the reusable part iiiVFW32.
- 5. Execute the Framework as Designer.
- 6. Open the *Properties* dialog for the Reports business object.
- 7. Select the *Commands Enabled* tab.
- 8. Select the *Monthly* action and plug in the *Reports Monthly* command handler, iiiVFW32. Use the *Find* dialog, which will plug in using the Identifier.
- 9. *Save and Restart* the Framework.
- 10. Select the *Reports* business object. The *Weekly* command handler will be displayed as this is a default command. Right click on the *Weekly* tab to display the context menu and select the *Monthly* command handler.

A list of all employees should be displayed.

11. Hold down the Control key and click on a number of employees in the Monthly list view.

The *Selected Employees* form should be displayed and show the fullname of all currently selected employees in its list view.

12. Change the selected employees on the Monthly command handler and note that the *Selected Employees* viewer is dynamically refreshed.

## Summary

## **Important Observations**

- Unlike fields which, once defined, are persistent for the lifespan of a component, components defined within a routine only exist for the duration of the routine. This is a very useful technique that saves you having to remember to reset variables each time a routine is run.
- Defining methods such as *SelectionChanged* adds much greater clarity to your code. Furthermore, the method can be invoked by other components.

## **Tips & Techniques**

- List Collections can collect any type (or Class) of Visual LANSA object.
- You would typically use Keyed Collections of Style Collection when you are collecting objects.
- You can pass a reference to an entire collection to another form or reusable component.

## What You Should Know

• How to pass a reference to an entire collection.

## VFW110 – Simple Drag and Drop

## Introduction

Drag and Drop is a common and useful feature in Windows applications. You should already be familiar with the concepts of drag and drop from your experience using the Visual LANSA development environment. For example, you drag and drop controls onto a form.

Most users will be familiar with drag and drop as it is an integral part of the Windows operating system. Drag and Drop is a practical way to copy or move an item from one context area to another. The copied or moved item can be virtually anything.

Drag and Drop includes these steps:

- 1. Select the item to 'drag' by pressing a mouse button down (usually the left button) on the item.
- 2. While the mouse button is held down, move the mouse to the target destination area.
- 3. Release the mouse button so that the item to be copied/moved is "dropped" in the target area.

There are four different events that can be involved in a drag and drop operation:

- StartDrag
- ragOver
- DragDrop
- EndDrag.

Note: A Drag and Drop operation may not require all of these events.

## Objective

To build a command handler for the *Departments* business object which enables Sections to be transferred to another Department, using drag and drop. This is described as 'simple drag and drop' because the action occurs within a single form. This simplifies the steps required to implement the drag and drop action.

epartme	nt		Descripti	on	Address Line 1	Address Line 2	
ADM			ADMINIS	TRATOR DEP	r		
OUA B			INTERNA	AL AUDITING			
FLT			FLEET AD	DMINISTRATIO	ON		
GAC			GROUP A	ACCOUNTS			
E INF			INFORM	ATION SERVIC	ES		
LEG			LEGAL DE	EPARTMENT			
D MIS			MANAGE	MNT INFORM			
MKT			MADVET	INC DEPARTM			
			CALEC B	DISTRIBUTIO	N		
D TDIA			TRAVEL O	DISTRIBUTIO	IN .		
E IRVL			TRAVEL	DEPARTMENT			
l Depa	rtment : Tr	ansfer (FLT-FLEET ADMIN	ISTRATIC	INI			
1 Depa	rtment : Tr	ansfer (FLT-FLEET ADMIN	ISTRATIC	DN)			
🔲 Depa	rtment : Tr Is 📿 Tra	ansfer (FLT-FLEET ADMIN nsfer	ISTRATIC	DN)			
Depa	rtment : Tr Is C Tra	ansfer (FLT-FLEET ADMIN	ISTRATIO				
Depa	rtment : Tr Is C Tra Sectio	ansfer (FLT-FLEET ADMIN nsfer ns from here	ISTRATIO	Section	ns in FLT Depart	tment.	
Depa Detai	rtment : Ti Is C Tra Section Section	ansfer (FLT-FLEET ADMIN nsfer ns from here		Section	ns in FLT Depart	tment.	
Detai	Is C Tra	ansfer (FLT-FLEET ADMIN nsfer ns from here Section Full Description INTERNAL ADMIN SRV		Section Section	Section Description	tment.	
Depart Depart ADM ADM	Section . 01 02	INTERNAL ADMIN SECTION		Section Section 01 02	Section Description ADMINISTRATION PURCHASING	tment.	
Depart Depart ADM ADM	Is C Tra Section Section.	Insfer (FLT-FLEET ADMIN Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION		Section Section 01 02 03	Section Description ADMINISTRATION PURCHASING ACCOUNTING	tment.	
Depart Depart ADM ADM ADM	rtment : Tra Sectio Section . 01 02 03 04	Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depart Depart ADM ADM ADM ADM	rtment : Tra Sectio Section . 01 02 03 04 05 06	Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PURCHASING SECTIONES		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depart ADM ADM ADM ADM ADM ADM ADM	timent : 1 Sectio Sectio 01 02 03 04 05 06	ansfer (FLT-FLEET ADMIN nsfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX URUSC MARKETING		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depart ADM ADM ADM ADM ADM ADM ADM ADM	timent : 1 Sectio Sectio 01 02 03 04 05 06 09 01	INTERNAL ADMIN ACCOUNTING SECTION ACCOUNTING SECTION ACCOUNTING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX VEHICLE MAINTENANCE ADMINISTINATION		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depa Deta Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	tment : 1 Section . Section . 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 04 05 05 05 05 05 05 05 05 05 05	Ansfer (FLT-FLEET ADMIN Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX VEHICLE MAINTENANCE ADMINISTRATION PURCHARDIC		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depa Deta Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	timent : 1 Section . Section . 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 04 05 06 07 07 07 07 07 07 07 07 07 07	ansfer (FLT-FLEET ADMIN nsfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONIXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depart ADM ADM ADM ADM ADM ADM ADM ADM	timent : 1 Section . Section . 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 04 05 06 07 07 07 07 07 07 07 07 07 07	ansfer (FLT-FLEET ADMIN nsfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONICX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ADMINISTRATION		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depa Deta Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM AUD AUD AUD GAC	timent : 1 Section . 01 02 03 04 05 06 09 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 04 05 05 05 05 05 05 05 05 05 05	ansfer (FLT-FLEET ADMIN nsfer INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONICX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Detai Detai Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM AUD AUD GAC GAC GAC	timent : 1 Section . 01 02 03 04 05 06 09 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 04 05 06 09 01 02 03 01 02 03 04 05 06 09 01 02 03 04 05 06 09 01 02 03 03 04 05 06 03 03 04 05 06 03 03 04 03 03 03 04 03 03 03 04 03 03 03 04 03 03 03 04 03 03 03 04 03 03 03 03 04 03 03 03 03 03 04 03 03 03 03 04 03 03 03 03 03 03 03 04 03 03 03 03 03 03 03 03 03 03	Ansfer (FLT-FLEET ADMIN Insfer INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ADMINISTRATION PURCHASING ACCOUNTING	E	Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Detai Detai Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	tment : 1 Section . Section . 01 02 03 04 05 06 09 01 02 03 01 02 03 01 02 03 04 05 06 09 01 02 03 05 06 09 01 02 03 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 06 05 05 06 05 06 05 06 05 05 05 05 05 05 05 05 05 05	Ansfer (FLT-FLEET ADMIN Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ADMINISTRATION PURCHASING ACCOUNTING EDNANCEIAL CONTROL	E	Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depa Detai Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	thent : 1 Section . Section . 01 02 03 04 05 06 09 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 04 05 06 09 01 02 03 01 02 03 04 05 06 01 02 03 04 05 06 01 02 03 01 02 03 04 05 06 01 02 03 04 05 06 01 02 03 01 02 03 04 05 06 01 02 03 04 01 02 03 04 05 06 01 02 03 01 02 03 04 05 06 01 02 03 01 02 03 04 01 02 03 01 02 03 04 01 02 03 04 03 01 02 03 04 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 03 02 03 03 03 03 03 03 03 03 03 03	ansfer (FLT-FLEET ADMIN nsfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION	E	Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depa Detai Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	thent : 1 Section . O1 O2 O3 O4 O5 O6 O9 O1 O2 O3 O1 O2 O3 O1 O2 O3 O1 O2 O3 O4 O5 O6 O9 O1 O2 O3 O1 O2 O3 O4 O5 O6 O1 O2 O3 O4 O5 O6 O7 O7 O7 O7 O7 O7 O7 O7 O7 O7	ansfer (FLT-FLEET ADMIN Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONIXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION PURCHASING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION PURCHASING	E	Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Detai Detai Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	timent : 1 Section . 01 02 03 04 05 06 09 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 03 02 03 03 02 03 03 03 03 02 03 03 03 03 03 03 03 03 03 03	Ansfer (FLT-FLEET ADMIN Insfer INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION ACCOUNTING SECTION ACCOUNTING ALES & MARKETING MAINTENANCE PERSONNEL SECTIONXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION PURCHASING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING	E	Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	
Depa Detai Depart ADM ADM ADM ADM ADM ADM ADM ADM ADM ADM	tment : 1 Section . Section . 01 02 03 04 05 06 09 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 01 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 02 03 03 02 03 02 03 02 03 02 03 02 03 02 03 02 02 02 02 02 02 02 02 02 02	Ansfer (FLT-FLEET ADMIN Insfer Section Full Description INTERNAL ADMIN SRV PURCHASING SECTION ACCOUNTING SECTION SALES & MARKETING MAINTENANCE PERSONNEL SECTIONIXX VEHICLE MAINTENANCE ADMINISTRATION PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION PURCHASING ACCOUNTING FINANCIAL CONTROL ADMINISTRATION PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING PURCHASING ACCOUNTING ACCOUNTING PURCHASING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING ACCOUNTING AC		Section Section 01 02 03 04	Section Description ADMINISTRATION PURCHASING ACCOUNTING SECTION FOUR	tment.	

To achieve this objective you will complete the following steps:

Step 1. Create Transfer Section to Department Command HandlerStep 2. Add Logic to the Transfer Sections to Department Command Handler.Step 3. Test the Transfer Section to Department Command HandlerSummary

## **Before You Begin**

You must have completed exercises VFW030, VFW040 and VFW042.

# **Step 1. Create Transfer Section to Department Command Handler**

1. Create a new *Reusable Part / Panel*:

#### Name: iiiVFW34

#### Description: Transfer Section to Department

- 2. Give the reusable part an ancestor of VF_AC010
- 3. Using the *Design* tab, give the panel iiiVFW33 a *Vertical Split manager*.
- 4. Open the *Layout Helper* tab (*Home* ribbon / *Views*). Give the *Vertical Splitter* a *Divider Style* of **Raised**.
- 5. Change the *Name* of PANL_1 to **LEFT**, and change the *Name* of PANL_2 to **RIGHT**.
- 6. Select the **LEFT** panel. On the *Design ribbon*, *give the panel and Attachment manager*.
- 7. Drop a Panel component onto the top of the LEFT panel and change its *Name* to **TOP_LEFT**.
- 8. Drop a Panel onto the bottom of the LEFT panel and change its *Name* to **BOTTOM_LEFT**.
- 9. Adjust the height of TOP_LEFT and BOTTOM_LEFT as required.
- 10. Drop a List View into the center of the panel LEFT. Change its *Name* to **DRAG_FROM**.
- 11. Save your reusable part.
- 12. Select the **RIGHT** panel and give it an *Attachment* manager.
- 13. Drop a Panel onto the top of the RIGHT panel and change its *Name* to **TOP_RIGHT**.
- 14. Drop a Panel onto the bottom of the RIGHT panel and change its *Name* to **BOTTOM_RIGHT**
- 15. Adjust the height of TOP_RIGHT and BOTTOM_RIGHT as necessary.
- 16. Drop a List View into the center of the RIGHT panel and change its *Name* to **DRAG_TO**.
- 17. Save your reusable part.

1	IIVFW33 - Transfer Section to Department	x

- 18. Drop a Label onto the TOP_LEFT panel.
  - a. Change its Text to Drag Sections from Here
  - b. Give it a *VisualStyle* of **VS_LAREM**
  - c. Resize the Label as necessary.
- **Hint:** To quickly make the label fit the TOP_LEFT panel, give the label a *Height* of **22**.
- 19. Follow the same procedure as 19. (above) to add a label for 'Department Sections' to the TOP_RIGHT panel.
- 20. Locate the file SECTAB on the *Repository* tab. Drag the fields DEPTMENT, SECTION and SECDESC into the DRAG_FROM list view. Change the *WidthType* for the SECDESC column to **Remainder**.
- 21. Change the *DragStyle* of the DRAG_FROM list, to **Automatic**. This list will now allow drag and drop.
- 22. Add a Push Button to the BOTTOM_LEFT panel.
  - a. Change its *Caption* to **Refresh**
  - b. Change its *Name* to **PHBN_REFRESH**
  - c. Create a *Click* event for the button.
- 23. Drag the fields STD_CODE and STD_DESC and STD_CODEL into the **DRAG_TO** list view.
  - a. Make the *WidthType* for the STD_DESC column, to **Remainder**.
  - b. Change the *Caption* for the STD_CODE column to **Section**. Change *CaptionType* to **Caption**.

- c. Change the *Caption* for the STD_DESC column to **Section Description**. Change *CaptionType* to **Caption**.
- d. Change the column for STD_CODEL to *Visible* False.
- 24. Change the *DragStyle* for list DRAG_TO, to **Automatic**.
- 25. Add a Push Button to the BOTTOM_RIGHT panel.
  - a. Change its *Caption* to **Save to Section Table**
  - b. Change its *Name* to **PHBN_SAVE**.
  - c. Create a Click event for it.
- 26. Save your reusable part.

		IIIVFW33 - Transfer Sec	tion to Department
Dra	ng Sect	ions from Here	Department Sections
Depart	Section	Section Full Description	Section Section Description
<b>√∂</b> •A	AB	aAbBcCdDeEfFgGhHIIjJ	A. aAbBcCdDeEfFgGhHiIjJkKlumMnN
R	efresh		Save to Section Table

# **Step 2. Add Logic to the Transfer Sections to Department Command Handler.**

- 1. On the *Design* tab, select the DRAG_FROM list view and create an *Initialize* event.
- 2. Select the DRAG_TO list view and create *DragDrop* event.
- 3. Define a work field CURRDEPT referred to field DEPTMENT. This field will hold current department
- 3. Switch to the *Source* tab. Create an *uExecute* method routine, which redefines the ancestor method
  - a. Add code to retrieve the *Akey1* for the current instance list entry into field CURRDEPT. *Akey1* for the Departments instance list contains DEPTMENT.
  - b. Execute a subroutine called *BldDragTo*.

Your code should look like the following:

Mthroutine Name(uExecute) Options(*redefine) * Return Akey1 into work field CURRDEPT #avlistmanager.getCurrentInstance Akey1(#currdept) Execute Subroutine(BldDragTo) Endroutine

4. Create a *BldDragTo* subroutine to populate the DRAG_TO list view from the file SECTAB for the key CURRDEPT.

Your code should look like the following:

```
Subroutine Name(BldDragTo)
Clr_List Named(#DRAG_TO)
Select Fields(#section #secdesc) From_File(sectab) With_Key(#currdept)
#std_codel := #currdept
#std_code := #section
#std_desc := #secdesc
Add_Entry To_List(#DRAG_TO)
Endselect
Endroutine
```

5. Complete the *Initialize* event routine for the list view DRAG_FROM. This

needs to contain entries for all records in the file SECTAB, except those for the current department.

The *Refresh* button click event will also rebuild the DRAG_FROM list view.

- a. Create a subroutine named *BldDragFrm*.
- b. Execute the *BldDragFrm* subroutine from the DRAG_FROM *Initialize* event routine.

Your code should look like the following:

```
Evtroutine Handling(#DRAG_FROM.Initialize) Options(*NOCLEARMESSA
Execute Subroutine(BldDragFrm)
Endroutine
Subroutine Name(BldDragFrm)
Clr_List Named(#DRAG_FROM)
* add all records, except for current department
Select Fields(#DRAG_FROM) From_File(sectab)
If (#deptment *NE #currdept)
Add_Entry To_List(#DRAG_FROM)
Endif
Endselect
Endroutine
```

6. The drag / drop operation requires that the target of the drag / drop accepts the proposed drop operation. Usually this will require that the target carries out some kind of validation in its *DragOver* event. For example it may check that the payload object (which is not required for drag and drop within the same form, as here) is a valid object. In this case no validation is necessary and the DRAG_TO list view will simply return an *AcceptDrop* of True. Complete the *DragOver* event for DRAG_TO.

Your code should look like the following:

```
Evtroutine Handling(#DRAG_TO.DragOver) Options(*NOCLEARMESSAGI
#acceptdrop := true
Endroutine
```

7. The *DragDrop* event needs to process the copy or move operation. Once again in this case, the requirements are simplified because this is a drag / drop operation within the same form.

Evtroutine Handling(#DRAG_TO.DragDrop) Options(*NOCLEARMESSAG]

Selectlist Named(#DRAG_FROM) If (#DRAG_FROM.currentItem.selected = true) #std_codel := #deptment #std_code := #section #std_desc := #secdesc Add_Entry To_List(#DRAG_TO) Endif Endselect Endroutine

- 8. Complete the logic for the *Save* button *Click* event. This needs to process the entries in the list DRAG_TO, which contains two types of entry:
  - i. Entries for the current department. Their STD_CODEL column contains the current department code.
  - ii. Entries for the proposed section transfers to the current department. These have a STD_CODEL column containing their existing department code.

The requirement is to:

- a. Read all DRAG_TO entries and ignore those for the current department.
- b. Check if the proposed transfer section already exists for current department.
- c. For accepted transfers, retrieve all fields from their current SECTAB record.
- d. Insert a new record in SECTAB for current department and new section code.
- e. Issue a message for successful inserts.
- f. Delete the SECTAB record for the transferred section, using their previous department code.
- g. Delete their entry from the DRAG_FROM list view.
- h. Issue a message for each proposed transfer section which is rejected.

Your code should look like the following:

Evtroutine Handling(#PHBN_SAVE.Click) Selectlist Named(#DRAG_TO) #deptment := #std_codel #section := #std_code
```
#secdesc := #std desc
* Bypass entries for current department.
If (#deptment *NE #currdept)
* Check for this section in SECTAB for current department
Check_For In_File(sectab) With_Key(#currdept #section) Val_Error(*next)
If_Status Is_Not(*equalkey)
* get other data for section
Fetch Fields(*all) From_File(sectab) With_Key(#deptment #section)
* Reset department code to current
#deptment := #currdept
Insert Fields(*all) To_File(sectab) Val_Error(*next)
If_Status Is(*okay)
Message Msgtxt('Section ' + #section + ' transferred to department ' + #currdep
* Remove section from donor department and DRAG_FROM list
#deptment := #std codel
Delete From_File(sectab) With_Key(#deptment #section) Val_Error(*next)
Dlt_Entry From_List(#DRAG_FROM)
Endif
Else
Message Msgtxt('Section ' + #section + ' not transferred')
Dlt_Entry From_List(#DRAG_TO)
Endif
Endif
Endselect
Endroutine
```

9. Complete the Refresh push button *Click* event.

It should execute the *BldDragFrm* subroutine:

```
Evtroutine Handling(#PHBN_REFRESH.Click)
Execute Subroutine(BldDragFrm)
Endroutine
```

10. Compile your new command handler.

# **Step 3. Test the Transfer Section to Department Command Handler**

- 1. Execute the Framework as a Designer.
- 2. Open the properties dialog for the _Departments business object.
- 3. Select the *Commands Enabled* tab. Select the *Transfers* action and plug in the command handler, iiiVFW34.
- 4. Save and restart the Framework.
- 5. Select the Transfer command handler for a department. The departments AUD and FLT usually have fewer sections than ADM, so select one of these.
- 6. Test the drag and drop application with the following steps.
  - a. Drag a section from the drag from list, which exists in the drag to list. Click the *Save* button and confirm that the transfer is rejected with an error message. The new entry should be removed from the drag to list.
  - b. Drag a section which does not already exist into the drag to list and click the *Save* button. A message should confirm the transfer and the entry should be removed from the drag from list.
  - c. Drag two sections to the drag to list, one of which does not already exist in the current department. Click the Save button. Confirm that only the new section remains in the drag to list, a transfer message is issued and the rejected transfer section remains in the drag from list.

# Summary

## **Important Observations**

- The *DragOver* event occurs when a drag and drop operation is in process. You can use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. The mouse pointer position determines the target object that receives this event.
- The *DragDrop* event occurs when the mouse button is released. During this event, you would normally populate the target with the dragged data.

# **Tips & Techniques**

• Refer to the Feature Help (F2) for more details about the Drag and Drop properties and events.

# What I Should Know

- How to enable a control for Drag and Drop.
- How to use the DragOver and DragDrop events.
- The purpose of the AcceptDrop property.
- How to Drag and Drop items between two controls in the same form.

# VFW112 – Drag and Drop between Components

# Objective

To learn how to drag and drop items between components.

- To create a payload component to drag items between components.
- To create a Transfer Employee to Section command handler.
- To create a reusable part to receive the employee data transferred via the payload object



To achieve these objectives you will complete the following steps:

Step 1. Create Employee Payload Object

Step 2. Create Reusable Part Section Employees

Step 3. Create the Transfer Employees to Section Command Handler

Step 4. Complete the Section Employees Component

Step 5. Complete the Transfer Employees to Section Command Handler Summary

## **Before You Begin**

- Complete exercises VFW030, VFW040 and VFW042 before starting this exercise.
- You also should complete VFW110 Simple Drag and Drop before starting this exercise.

# **Step 1. Create Employee Payload Object**

In this step you create two components.

- A simple Employee Object to store properties for each employee.
- An Employee Payload object which contains a list collection of the Employee Object, containing the employees being transferred.
- 1. Create a new Reusable Part / Object:

### Name: iiiVFW35

### Description: Employee Object

2. Define properties uDepartment, uSection, uEmpNum and uFullname.

The properties should automatically set and get the corresponding field values.

Your code should look like the following:

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #PRIM_OBJT) Define_Pty Name(uDepartment) Get(*auto #deptment) Set(*auto #deptment) Define_Pty Name(uSection) Get(*auto #section) Set(*auto #section) Define_Pty Name(uEmpNum) Get(*auto #empno) Set(*auto #empno) Define_Pty Name(uFullname) Get(*auto #fullname) Set(*auto #fullname) End_Com

- 4. Compile iiiVFW35
- 5. Create a new *Reusable Part / Object*:

Name: iiiVFW36

Description: Employee Payload

- Define a list collection which collects the employee object, iiiVFW35: Define_Com Class(#prim_lcol<#iiivfw35>) Name(#Objects)
- 7. Define a property which passes a reference to the list collection (called #objects) when another component retrieves this property:

Define_Pty Name(pObjects) Get(*collection #Objects)

8. Define a method routine, *Add_Item*, with input parameters for the employee object properties:

Mthroutine Name(Add_Item) Define_Map For(*input) Class(#DEPTMENT) Name(#uDeptment) Define_Map For(*input) Class(#SECTION) Name(#uSection) Define_Map For(*INPUT) Class(#EMPNO) Name(#uEmployee) Define_Map For(*INPUT) Class(#FULLNAME) Name(#uFullname) Endroutine

- 9. Complete the *Add_Item* method routine.
  - a. Add code to define an instance of the employee object iiiVFW35, named New_Object.
  - b. Update the properties of New_Object using the passed parameters.
  - c. Insert New_Object to the list collection, named Objects.

Mthroutine Name(Add_Item) Define_Map For(*input) Class(#DEPTMENT) Name(#uDeptment) Define_Map For(*input) Class(#SECTION) Name(#uSection) Define_Map For(*INPUT) Class(#EMPNO) Name(#uEmployee) Define_Map For(*INPUT) Class(#FULLNAME) Name(#uFullname) Define_Com Class(#iiivfw35) Name(#New_Object) #New_Object.uDepartment := #uDeptment #New_Object.uDepartment := #uEmployee #New_Object.UEmpNum := #uEmployee #New_Object.UFullname := #uFullname #Objects.Insert( #New_Object ) Endroutine

10. Compile iiiVFW36.

# **Step 2. Create Reusable Part Section Employees**

This component will be displayed on the right hand side of the Transfer command handler. It displays the employees belonging to the section selected in the instance list. The drag and drop operation allows an employee(s) to be transferred to the Section from another section. A Save button enables the changes to be saved to the file SECTAB.

1. Create a new *Reusable Part / Panel*:

Name: iiiVFW37

Description: Section Employees.

- 2. On the *Design* tab, use *New Layout* to give iiiVFW37 an *Attachment manager*.
- 3. Drag and drop a List View component to the center of the Panel. Change the List View *Nam*e to **DRAG_TO**.
- 4. Add fields DEPTMENT, SECTION, EMPNO, FULLNAME and STD_CODEL to the List View DRAG_TO.
- 5. Make the STD_CODEL column *Visible*, **False**.
- 6. Change the list view column headings to suitable short values. For example: Dept., Sect., Emp No. and Full Name. Change each column's *CaptionType* to **Caption**.

Change the Full Name column's *WidthType* = **Remainder**.

- 7. Save the reusable part.
- 8. The Section Employees component is not a command handler, and therefore cannot communicate directly with the Framework. The *Transfer Employee* command handler (which you will create in the next step) will retrieve department and section codes for the current instance list entry. It will then set properties in iiiVFW37 to enable this component to populate the DRAG_TO list view with employees for the current instance list entry.
  - a. Define the following properties in iiiVFW37:

Define_Pty Name(uDepartment) Set(*auto #deptment) Define_Pty Name(uSection) Set(BuildList)

b. Create a property routine *BuildList*, to clear and populate the DRAG_TO list with employees.

Your code should look like the following:

```
Ptyroutine Name(BuildList)
Define_Map For(*input) Class(#section) Name(#i_section)
#std_codel := *blanks
Clr_List Named(#DRAG_TO)
Select Fields(#deptment #section #empno #surname #givename) From_File(ps
#fullname := #surname + ', ' + #givename
Add_Entry To_List(#DRAG_TO)
Endselect
Endroutine
```

c. Review the BuildList property routine.

Section code (SECTION) is passed into this routine as *i_section*.

The STD_CODEL column is set to blanks. Entries with a blank STD_CODEL will be recognized as existing employees for the section.

The list DRAG_TO is cleared and populated with all employees for the department, section using logical file PSLMST1.

9. Compile iiiVFW37 in its current form and leave it open in the editor. You will add additional logic for drag and drop in a later step.

# **Step 3. Create the Transfer Employees to Section Command Handler**

1. Create a new Reusable Part / Panel:

Name: iiiVFW38

#### Description: Transfer Employees to Section CH

- 2. To define the user interface, replace the code for iiiVFW38 with the code supplied in VFW112 Appendix. You created a similar interface design in the VFW110 Simple Drag and Drop.
- 3. Change the supplied code so that the DRAG_TO component uses your initials for iiiVFW37.
- 4. Switch to the *Design* view. Your design should look like the following:



A vertical splitter component divides the panel. The DRAG_FROM list on the left will contain employees for all department / sections excluding the department/section selected in the instance list. That is, all employees that could be transferred to the current department/section. The DRAG_FROM list view has a *DragStyle* property of **Automatic**.

The right side contains the Section Employees component, iiiVFW37. Employees may be dragged into this component's list and then saved by updating the employee file.

5. Create work fields for current department and section values:

Define Field(#currdept) Reffld(#deptment) Define Field(#currsec) Reffld(#section) 6. Create an *uExecute* method routine:

```
Mthroutine Name(uExecute) Options(*redefine)
#avlistmanager.getCurrentInstance Akey1(#deptment) Akey2(#section)
#currdept := #deptment
#currsec := #section
#DRAG_TO.uDepartment := #deptment
#DRAG_TO.usection := #section
Execute Subroutine(BuildEmps)
Endroutine
```

Review the *uExecute* logic:

Department and section codes are retrieved from the instance list, as *Akey1* and *AKey2* columns.

DRAG_TO is the name of the iiiVFW37 component.

The *BuildEmps* subroutine does not yet exist.

7. Create the *BuildEmps* subroutine:

```
Subroutine Name(BuildEmps)
Clr_List Named(#DRAG_FROM)
Select Fields(#deptment #section #empno #surname #givename) From_File(ps
If ((#currdept *NE #deptment) *And (#currsec *NE #section))
#fullname := #surname + ', ' + #givename
Add_Entry To_List(#DRAG_FROM)
Endif
Endselect
Endroutine
```

All entries from PSLMST1 are added to the list view DRAG_FROM, except employees for the instance list current department / section.

- 8. On the *Design* view, select the DRAG_FROM list view on the left panel and create *StartDrag* and *EndDrag* event routines.
- 9. Give the *StartDrag* event routine *Payload*, *Continue* and *DragList* parameters as shown.

Evtroutine Handling(#DRAG_FROM.StartDrag) Options(*NOCLEARMESS/ Endroutine **Note:** The *AutoComplete* prompter will complete these keywords as you type.

10. In the StartDrag routine, define a *Payload_Employee* component with a class of iiiVFW36 which contains a list collection of Employee Object (iiiVFW35).

Evtroutine Handling(#DRAG_FROM.StartDrag) Options(*NOCLEARMESS/ Define_Com Class(#iiivfw36) Name(#Payload_Employee) Endroutine

11. The *StartDrag* event is triggered when the user selects an entry and drags the mouse with the left mouse button held down.

This routine needs to add selected entries from the DRAG_FROM list view to the Payload_Employee component. That is, into the payload object.

The 'add to payload list collection' is performed by invoking the *Add_Item* method passing the parameters required.

Add the following code to the *StartDrag* event routine to achieve this:

Selectlist Named(#DRAG_FROM)

Continue If(*Not #DRAG_FROM.currentItem.selected)

#Payload_Employee.add_item Udeptment(#deptment) Usection(#section) Uen
Endselect

12. Having populated the *Payload_Employee* object, the following is required:

- a. A reference to Payload_Object is passed as Payload.
- b. The *DragList* parameter is set to **selection**, meaning that the drag image will be the list view's selected items.
- c. The *Continue* parameter is set to true.

Add the following code to achieve this:

```
Set_Ref Com(#Payload) To(#Payload_Employee)
Set Com(#draglist) Dragliststyle(selection)
#continue := true
```

13. Add *Payload* and *DragResult* parameters to the *EndDrag* routine.

Evtroutine Handling(#DRAG_FROM.EndDrag) Options(*NOCLEARMESSA Endroutine

14. Define the *Payload_Employee* object with a class of iiiVFW36 and Reference of *dynamic.

Create the *Payload_Employee* object with a Set_Ref to the parameter value #Payload.

Implement this with the following code:

Define_Com Class(#iiivfw36) Name(#Payload_Employee) Reference(*dynam Set_Ref Com(#payload_employee) To(*dynamic #payload)

15. If the *DragResult* parameter is ACCEPTED, delete currently selected entries from the DRAG_FROM list view.

Add the following code to implement this:

If (#dragresult = ACCEPTED) Selectlist Named(#DRAG_FROM) Continue If(*Not #DRAG_FROM.currentItem.selected) Dlt_Entry From_List(#DRAG_FROM) Endselect Endif

16. Compile the *Transfer Employees to Section* command handler (iiiVFW38). You will add additional logic for the push buttons in a later step.

# **Step 4. Complete the Section Employees Component**

Switch to the Section Employees component, iiiVFW37.

- 1. On the *Design* tab, select the DRAG_TO list view and create *DragOver* and *EndDrag* event routines.
- 2. Complete the *Drag_Over* event by defining *Payload* and *AcceptDrop* parameters.

Set AcceptDrop to true if the Payload_Employee object is component class iiiVFW36.

Your code should look like the following:

Evtroutine Handling(#DRAG_TO.DragOver) Options(*NOCLEARMESSAGI Endroutine

3. Define *DragResult* and *Payload* parameters for the *DragDrop* routine.

As for the *DragOver* routine, define a dynamic Payload_Employee object with a class of iiiVFW36.

Create Payload_Employee using *Set_Ref* to Payload.

Your code should look like the following:

Evtroutine Handling(#DRAG_TO.DragDrop) Options(*NOCLEARMESSAGI Define_Com Class(#iiiVFW36) Name(#Payload_Employee) Reference(*dy Set_Ref Com(#Payload_Employee) To(*dynamic #Payload) Endroutine

4. Retrieve each payload entry using a For / Endfor loop and add an entry to the DRAG_TO list.

Set the STD_CODEL column to the payload department code.

Your code should look like the following:

```
Define #CurrDept Reffld(#Deptment)
#currdept := #deptment
For Each(#object) In(#Payload_Employee.pObjects)
#deptment := #currdept
#std_codel := #object.udepartment
#empno := #object.uempNum
#fullname := #object.ufullname
```

Add_Entry To_List(#DRAG_TO) Endfor

5. Create a *SaveChanges* method routine to be invoked via the *Transfer Employees to Section* command handler *Save* button.

A transfer will be implemented by updating department and section code fields in the employee record.

Transferred employees have a non blank STD_CODEL column.

Issue a message if an update is not successful.

Your code should look like the following:

Mthroutine Name(SaveChanges) Selectlist Named(#DRAG_TO) If (#std_codel *NE *blanks) Update Fields(#deptment #section) In_File(pslmst) With_Key(#empno) Val_E If_Status Is_Not(*okay) Message Msgtxt('Transfer employee ' + #empno + 'to ' + #deptment + ' failed') Endif Endif Endif Endselect Endroutine

6. Compile Section Employees, iiiVFW37.

# **Step 5. Complete the Transfer Employees to Section Command Handler**

- 1. Switch to iiiVFW38 in the editor.
- 2. Create *Click* events for the PHBN_REFRESH and PHBN_SAVE buttons.
- 3. The *Refresh* push button click event should rebuild the DRAG_FROM list. Evtroutine Handling(#PHBN_REFRESH.Click) Execute Subroutine(BuildEmps) Endroutine
- 4. The *Save* push button click event should invoke the *SaveChanges* method in the DRAG_TO component (iiiVFW37).

Evtroutine Handling(#PHBN_SAVE.Click) #DRAG_TO.SaveChanges Endroutine

- 5. Compile the *Transfer Employees to Section* command handler.
- 6. Execute the Framework as Designer.
- 7. Double click on a section in the instance list to open its *Properties* dialog.
- 8. Select the *Commands Enabled* tab, select the *Transfer* command and plug in iiiVFW38. Use the *Find* dialog, which plugs in using the Identifier.
- 9. Save and Restart the Framework.
- 10. Test the Transfer Employees command handler for a Section.
  - a. Select the Departments business object, expand ADM and select a section.
  - b. Select the Transfers command handler. The left hand (DRAG_FROM) List contains all employees for all department / sections except the currently selected department / section.
  - c. The right hand list (DRAG_TO) contains employees in the selected department / section.
  - d. Select one or more employees in DRAG_FROM and drop them on DRAG_LIST. The employees are shown in DRAG_TO with their department and section currently selected in the instance list. These

employees have been removed from the DRAG_FROM list. Note that the drag image is the selected list items.

e. Click the Save to Employee File button to update the transferred employees, changing their department and section codes to the selected department and section.

## Summary

## **Important Observations**

- The default value of the DragStyle property is none. This means that the component will NOT support the initiation of a drag operation.
- The StartDrag event occurs when the mouse button is pressed over a component and the mouse is moved. It is typically used to store away the data being dragged. As such, it might not be required when the information is dropped in the same form.
- The DragOver event occurs when a Drag and Drop operation is in progress. You would decide whether to accept the drop operation during the DragOver event.
- The DragDrop event occurs when the mouse button is released. During this event you would normally populate the target with the dragged data.
- At the end of the DragDrop event the EndDrag occurs. It is a notification to the source that the Drag has finished. The routine would typically contain code which is related to the source component. For example, set DragResult to ACCPETED and delete selected entries from the DRAG_FROM list.

# **Tips & Techniques**

• Use a payload object as a temporary "storage" area when the data being dragged is not available in the component where it is to be dropped.

# What I Should Know

- How to Drag and Drop multiple items between two controls in different forms.
- How to use the EndDrag routine.
- How to use the DragResult property.
- How to copy or move items using drag and drop.

# VFW112 – Appendix

To define the user interface, use the following code to replace the code for iiiVFW38. This code is supplied to save time:

Function Options(*DIRECT)

Begin_Com Role(*EXTENDS #VF_AC010) Height(270) Layoutmanager(#SF Define_Com Class(#PRIM_SPLM) Name(#SPLM_1) Dividerstyle(Raised) Or Define Com Class(#PRIM PANL) Name(#LEFT) Displayposition(1) Height( Define Com Class(#PRIM PANL) Name(#RIGHT) Displayposition(2) Heigh Define_Com Class(#PRIM_SPLI) Name(#SPLI_1) Manage(#LEFT) Parent(#SPLI_1) Manage(#SPLI_1) Manage(#S Define_Com Class(#PRIM_SPLI) Name(#SPLI_2) Manage(#RIGHT) Parent( Define Com Class(#PRIM ATLM) Name(#ATLM 1) Define Com Class(#PRIM PANL) Name(#LEFT TOP) Displayposition(1) H Define_Com Class(#PRIM_ATLI) Name(#ATLI_1) Attachment(Top) Manage Define_Com Class(#PRIM_PANL) Name(#LEFT_BOTTOM) Displayposition Define_Com Class(#PRIM_ATLI) Name(#ATLI_2) Attachment(Bottom) Man Define_Com Class(#PRIM_LTVW) Name(#DRAG_FROM) Componentversic Define_Com Class(#PRIM_ATLI) Name(#ATLI_3) Attachment(Center) Mana Define_Com Class(#PRIM_PANL) Name(#RIGHT_TOP) Displayposition(2) Define_Com Class(#PRIM_ATLI) Name(#ATLI_4) Attachment(Top) Manage Define Com Class(#PRIM PANL) Name(#RIGHT BOTTOM) Displayposition Define_Com Class(#PRIM_ATLI) Name(#ATLI_5) Attachment(Bottom) Man Define_Com Class(#PRIM_LABL) Name(#LABL_1) Caption('Drag Employe Define_Com Class(#PRIM_PHBN) Name(#PHBN_REFRESH) Caption('Refr Define_Com Class(#PRIM_PHBN) Name(#PHBN_SAVE) Caption('Save to E Define_Com Class(#PRIM_LABL) Name(#LABL_2) Caption('Section Emplo Define_Com Class(#PRIM_LVCL) Name(#LVCL_1) Caption('Dept.') Caption Define_Com Class(#PRIM_LVCL) Name(#LVCL_2) Caption('Sect.') Captiont Define_Com Class(#PRIM_LVCL) Name(#LVCL_3) Caption('Number') Capti Define_Com Class(#PRIM_LVCL) Name(#LVCL_4) Caption('Full Name') Ca * Change the following line using your initials Define_Com Class(#IIIVFW38) Name(#DRAG_TO) Height(197) Parent(#RIC Define_Com Class(#PRIM_ATLI) Name(#ATLI_7) Attachment(Center) Paren

Define_Com Class(#PRIM_ATLI) Name(#ATLI_7) Attachment(Center) Paren Define_Com Class(#PRIM_ATLI) Name(#ATLI_6) Attachment(Center) Mana End_Com

# VFW120 – Using Hidden Commands

# **Hidden Command Handler Anatomy**

- Hidden commands are run in the same way as other commands but do not appear on tabs or in separate windows and are hidden from the user
- Are used to perform non-visual tasks.
- For Windows applications hidden commands are reusable parts with an ancestor of VF_AC020.
- Have most non-visual VL Framework and instance list services available to them.

Structurally, Hidden Command Handlers for Windows applications are similar to Windows Command Handlers with these important differences:

- They extend the base class #VF_AC020.
- They don't use the optional method *uInitialize*.
- They don't use the optional method *uTerminate*.
- They don't listen to events.

When using Hidden Command Handlers it is important to remember that they:

- Should always have the *Default Command* option set to NEVER for instance level commands or NO for business object level commands.
- Should never be used with the *Hide All Other Command Tabs* option set.
- Should never be attached to *RAMP Destination screens*.
- Should never attempt to display information to the user or interact with the user. Hidden means hidden

# Objectives

- To create a demonstration hidden command, 'Reverse', which will act on the current instance list entry and reverse the employee full name string.
- This is a quick and simple example to demonstrate the principle.

To achieve this objective you will complete the following steps:

Step 1. Add Reverse as a Hidden Command for Employees

Step 2. Create the Reverse Command

Step 3. Plugin and Test Reverse Command

Summary

# **Before You Begin**

Complete exercises VFW030, VFW040 and VFW042.

# Step 1. Add Reverse as a Hidden Command for Employees

- 1. Open the *Properties* dialog for Employees and select the *Commands Enabled* tab.
- 2. Click the *Command Definitions* button.



- 3. Click the *New* button to define a new command.
- 4. Define the Reverse command as follows:

Caption	Reverse
Hint	Reverses Employee Full Name

User Object Name/Type REVERSE

**Note:** The *User Object Name/Type* is initially given a unique identifier (GUID) which is generated for all new framework objects. You can change this, provided the value is unique. The *Verify Name* button enables you to check that your name is unique within the framework.

- 5. Give the Reverse command any suitable icon and bitmap. Select the same image for each, so that the image will appear on tool bars and menus for the Reverse command.
- 6. Click the *Close* button.
- 7. Drag the Reverse command into the *Commands Enabled* column for the Employees business object.
- 8. Close the dialog which prompts you to save and restart your Framework and prevent it being re-displayed.
- 9. Select the checkbox to make the Reverse command *Execute as a hidden command*
- 10. Give the Reverse command a *Default Command* value of **Never.**

11. *Save and Restart* the Framework.

## Step 2. Create the Reverse Command

1. Create a new *Reusable Part / Panel*:

Name: iiiVFW41

## Description: Reverse Hidden Command.

2. Replace its code with the following:

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #VF_AC020)

* Simple Field and Group Definitions

Define Field(#REVSD) Reffld(#VF_ELBOOL)

Def_Cond Name(*REVSD) Cond('#REVSD *EQ TRUE')

* Handle Command Execution

Mthroutine Name(uExecute) Options(*REDEFINE)

* Do any execution logic defined in the ancestor

Invoke Method(#Com_Ancestor.uExecute)

* Get the Employee number of the current instance

Invoke Method(#avListManager.GetCurrentInstance) Akey1(#EMPNO) Acolu

* Fetch information from the main file to fill in the header fields on the form Fetch Fields(#SURNAME #GIVENAME) From File(PSLMST) With Key(#E

* Put the names together in the reverse order

If Cond(*REVSD)

* Put the names together Given name first

Change Field(#UF_VisID2) To(#GIVENAME)

Use Builtin(BConcat) With_Args(#UF_VisID2 #SURNAME) To_Get(#UF_Vi * Set the reversed flag

Change Field(#REVSD) To(FALSE)

Else

* Put the names together Surname first

Change Field(#UF_VisID2) To(#SURNAME)

Use Builtin(BConcat) With_Args(#UF_VisID2 #GIVENAME) To_Get(#UF_V

* Set the reversed flag

Change Field(#REVSD) To(TRUE)

Endif

* Update the name (Visual ID 2) to the instance list Invoke Method(#avListManager.UpdateListEntryData) Akey1(#EMPNO) Visu Endroutine End_Com

- 3. Compile your reusable part and review the code:
  - a. All hidden commands must have an ancestor of VF_AC020.
  - b. The *uExecute* method, retrieves *AKey1* and *AColumn3* for the current instance list entry. Note that you may need to change this column number if your instance list already uses it to hold a different value.
  - c. Fetches surname and givename for this employee number.
  - d. Tests the value of the condition *REVSD which is checks if the value returned from AColumn3 is TRUE.
  - e. Depending on *REVSD, VisualID2 is updated, with Surname first or Given Name first.
  - f. Updates the current list entry, columns VisualID2 and AColumn3.

## Step 3. Plugin and Test Reverse Command

- 1. Open the *Properties* dialog for Employees. Select the *Commands Enabled* tab.
- 2. Plug in the reusable iiiVFW41 for the Reverse command. Use the *Find* dialog, so that the plug in uses the Identifier name.
- 3. *Save and Restart* the Framework.
- 4. Test the Reverse command for the Employees business object.

**Note:** Initially, AColumn3 is blank. In this case, the hidden command sets VisualID2 to surname, given name. This is the same as its initial appearance so no change will be observed. Selecting an entry a second time will change VisualID2 to the form given name, surname and the change is obvious.

# Summary

# **Important Observations**

- A hidden command may perform any processing you require, but may not use the optional methods, *uInitialize* and *uTerminate*.
- Hidden commands do not listen for events.

# **Tips & Techniques**

- Hidden commands should be set to *Default Command*, NEVER.
- Refer to the *Visual LANSA Framework Guide* for further details about using hidden commands.

# What You Should Know

• How to implement a hidden command.

# VFW122 – Launching a VLF Window

# **Programatically Creating and Managing Windows**

Your programs can create and manage windows by calling the method avShowWindow in the Framework manager.

For example:

• Open a whole new instance of the Framework in another window named MYWINDOW:

#AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(FRAM

- Open the DEMONSTRATION application: #AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(APF
- Open an application view named HR: #AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(VIE
- Open business objects Organizations and Resources in two independent windows:

#AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(BU #AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(BU ofType(DEM_ORG_SEC_EMP) WindowName(DEMO_SECTION)

Remember that the *OfType(DETAILS)* names you specify on calls to *avShowWindow* are the *User Object Name / Type* values specified on the *Identification* tab of the properties of the respective object:

User Object Name / Type DETAILS Verify Name

# Using the avShowWindow method

When the *avShowWindow* method is invoked, it tests whether a window with the name specified exists.

If the named window already exists, it is activated (ie: restored from being minimized, if required, and brought to the front of all the Framework windows).

If the named window does not exist, it is created.

Then in both cases:

- The window's open information properties and object reference are updated with anything you supplied (see *Window Opening Information* in the *Visual LANSA Frameworks guide*.).
- A switch operation is performed inside the window to any application or business you may have nominated.

In simple terms, you are saying to the Framework "Display a window with this name and pass this information into it, then cause it to switch to this application or this business object".

When you create a new window or switch an existing window it happens asynchronously to your program. So if you use *avShowWindow* and then immediately enumerate all open windows you will not find the window you just created (yet).

# The uShowWindowCompleted Method

This method should be placed in a command handler which needs to "wake up" and perform some processing when requested by a command handler in the main VLF window:

For example a command handler in a window named EMP_WINDOW contains:

```
Mthroutine Name(uShowWindowCompleted) Options(*Redefine)
#EMPNO := #Com_Owner.avCurrentWindow.OpenInfo<1>
Fetch Fields(#XG_HEAD) From_File(PSLMST) With_Key(#EMPNO)
Endroutine
```

The method *uShowWindowCompleted* is run when another command handler performs:

#AvFrameworkManager.avShowWindow WindowName(EMP_WINDOW)

The methods *avShowWindow* and *uShowWindowCompleted* should be considered as a "pair".

# Window Names

You may have noticed from the preceding examples that windows have symbolic names. Here are some things you should know about window names:

- The names ALL, MAIN and CURRENT are reserved.
- When an end-user opens a window it is automatically assigned a unique name that starts with USER_. Do not rely on USER_ window names being

the same from Framework signon to signon or from Framework version to version.

- Names are not end-user visible. They are programmatic names, case insensitive and may be up to 256 characters long. Being case insensitive means they are often uppercased, so using just 'A' -> 'Z' and '0' -> '9' is advisable.
- Window names are uniquely scoped and only addressable within a Framework process (ie: a LANSA X_RUN.EXE process). This means that if you start multiple X_RUN.EXE processes they can each contain a unique window named TESTWINDOW. An operation that involves signalling or switching window TESTWINDOW only refers to it within the current X_RUN.EXE process. No intra-process windows operations are currently provided.

# **Application Settings**

The Properties for a Framework, an Application and a Business Object enable the designer to determine whether each object can be opened in a new window, *Never, Manually, Automatically* or *Automatically or Manually*:

entification Bitmaps and Icons	Startup	Visual Styles	Commands Enabled	Command	Display Help Abo
Caption	ii HR Application			(ENG)	
Hint:					(ENG)
Sequence:	1				
Internal Identifier:	47870EC	1F2954BB89A	A5C9C8B81CFF8A		
Unique Identifier:	83				
User Object Name / Type	III_HR_APPLICATION			Verify Name	
Restricted Access					
Contains Favorites					
Allow this Object to be Opened in a	New Wind	low	Manually		
Last Changed	20120210	0-085532-JIVO	RY Manually		
			Automatically Automatically or M	anually	

**Note:** The *Automatic* setting will **always** open the object in a new window when it is selected in the *Navigation* panel. It therefore means *always open this object in a new window*.

The setting applies to that level only. So a setting of *Never* at the Framework level, means the whole framework cannot be opened in a new window, but an

application or business object can be opened in a new window, if this is allowed for the object.

# Objective

To extend the *Employee Images for a Section* command handler to display employee details by opening a new Framework window.

Departments				×		
Eile Edit View Actions Help	Windows (Framework) (Administratio	n )				
🔁 📄 🥐 🛛	ā 🙆 🚨 🖾 🗟	) 💿 💿 🗎 🖱	ck Find	Employees:2 Eile Edit View Help V	Windows	
Other Windows - Broploy	ees:2	es Organization bookings Spool Hies		🚱 🗔 🙎	🎯 📄 🖉	🔺 👌 🔞
Departments	1			New About Resource	es Organizationspool mies i vew window	Queues Print Exit
E C Favorites				Other Windows - )00( Depa	artments (Main Window)	
HR Demo Application		Description Address Line 1	Address Line 1	Employees:2		
Generation     Departments     Employees	Esctions     Internal Admin SRV     PURCHASING SECTION	01 125 Main St, 02 123 Pacific Hichway,	Blacktown North Sydney	Sy Name 😥 By Locatio	on X	
Reports     II Transport Application     II HP Application		03 257 Canterlu rv R	CANTERRI IDV	REC	DFORD A:	1509 REDFORD ROBERT AD
Marcus Instance Example     Programming Techniques	Section : Images (INTERNAL ADMI	N SRV-01)		<	•	III F
🗄 🍓 Administration	Details S Employees 1 Images	Picture C Transfer		🎒 Employee : Details (A1	509-REDFORD, ROBERT)	•
Smothe Smiths Douglas Mrs brick Rediord			🗖 Detais 📓 Brief Notes 🚺 Images 💊 Notes 🚔 Skills 2 🍤 Skills			
		Open Employee	es Window	Employee Number	A1509	Business Phone Save
		Show Employee	Details	Employee Same	REDFORD	Departments Back to Departments
				Employee Given Name(s)	ROBERT	Section Code
				Street No and Name	122 Arthur Street	Employee Salar
🗉 🖬 📓	Messages Rea	dy Local ENG JIVORY12	22/08/12 9	Suburb or Town	North Sydney	Start Date (DD
				State and Country	NSW Australia	Termination Da
				Post / Zip Code	2060	
				Home Phone Number	5559966	
					Massage Baarly Local	FNG 10/00/12 22/00/12 9-04

To meet this objective you will complete the following steps:

Step 1. Extend Pop-Up Menu in Employee Image Object

Step 2. Enhance Employee Images for Section Command Handler

Step 3. Change Employee Details Command Handler

Step 4. Ensure Details displayed for first Employee Summary

# **Before You Begin**

Complete exercises VFW030, VFW040, VFW042, VFW100 and VFW104 before starting this exercise.

In exercise VFW104, you implemented *Sections* as a business object which is visible in the *Navigation* panel, with a *By Department* filter to populate the instance list. In this exercise you will add the ability to open *Employee Details* in a new window from the *Employee Images for a Section* command handler, working with the Sections business object itself. This will enable a 'switch back' to be implemented from *Employee Details* to *Employee Images for Section* in the main VLF window.

# Step 1. Extend Pop-Up Menu in Employee Image Object

- 1. Open iiiVFW29 in the editor
- 2. In the *Design* view, right click on the image component and select *Edit Popup Menu*.

Design Sou	rce   Repository Details   Reposi
IIIVFW28	A - E ×
	Image: IMGE_1
	Delete Component Copy Component Cut Component
	Edit Popup Menu Detach Popun Menu

The Popup menu is displayed at the top of the *Design* view.

3. Move the cursor to right hand side of the menu item text and press *Enter* to add a new menu item:



- 4. Define the new menu item as **Open Employee Details Window**. Note that the new menu item is MITM_2.
- 5. Create a *Click* event for MITM_2.
- 6. Switch to the *Source* tab.

- 7. Define an event uOpenEmpWindow, which will pass employee number: Define_Evt Name(uOpenEmpWindow) Define_Map For(*input) Class(#empno) Name(#uEmpNum)
- 8. Make the *Click* event for MITM_2, signal the *uOpenEmpWindow* event:

Evtroutine Handling(#MITM_2.Click) Options(*NOCLEARMESSAGES *NC Signal Event(uOpenEmpWindow) uEmpNum(#Empno) Endroutine

9. Recompile the new version of iiiVFW29

# **Step 2. Enhance Employee Images for Section Command Handler**

In this step you will change iiiVFW28 to handle the new event *uOpenEmpWindow* signalled by iiiVFW29.

At present the Employee Images for Section CH handles the *uShowEmpDetails* event by using *avSwitch* to open the Employee Details CH within the same Framework window.

The Employee Images for Section CH will handle the *uShowWindow* event by invoking the *avShowWindow* method to open the Employee Details command handler in a new Framework window.

In the next step you will change the Employee Details CH (iiiVFW06) to process the *avShowWindow* request.

- 1. Open *iiiVFW28 Employee Images for Section* in the editor
- 2. Add an event routine for *ImageCollection*<>.uOpenEmpWindow

Evtroutine Handling(#ImageCollection<>.uOpenEmpWindow) uEmpNum(#I_ Endroutine

This will execute when the pop-up menu option *Open Employee Details Window* is selected on an employee image.

3. This routine will invoke the *avShowWindow* method for the window name III_EMP.

The routine must first declare a temporary reference to the framework's *User Interface to multiple Windows* component, VF_SY154.

The Framework *avWindow* property returns a reference for a requested window name. New code is highlighted in red.

Evtroutine Handling(#ImageCollection<>.uOpenEmpWindow) uEmpNum(#i_ * define a temporary class #VF_SY154 reference

Define_Com Class(#vf_sy154) Name(#window) Reference(*dynamic) * Ask the framework manager to locate a window by name and return a r #window <= #avFRAMEWORKMANAGER.avWindow<'III_EMP'> Endroutine

4. If this reference is not *null, invoke *avShowWindow* for the business object EMPLOYEES to execute DETAILS, in window name III_EMP.

Evtroutine Handling(#ImageCollection<>.uOpenEmpWindow) uEmpNum(#i_ * define a temporary class #VF_SY154 reference

Define_Com Class(#vf_sy154) Name(#window) Reference(*dynamic)

* Ask the framework manager to locate a window by name and return a referer #Window <= #AvFrameworkManager.avWindow<'III_EMP'>

If (#Window *IsNot *Null)

Message Msgtxt('Window III_EMP is already open')

* Display employee details in III_EMP window

#empno := #i_empnum

#AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(BUS] Else

* Open a new window

#AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(BUS] Endif

Endroutine

- 5. Review the above code:
  - A temporary reference to VF_SY154 is defined.
  - Check if window III_EMP exists and then: set EMPNO from passed value I_EMPNUM.
  - Invoke avShowWindow for business object EMPLOYEES, window III_EMP, command handler DETAILS, passing EMPNO as open information 1. Else

use avShowWindow to open a new window.

6. Compile iiiVFW28.

# **Step 3. Change Employee Details Command Handler**

In this step you will change the *Employee Details* command handler, iiiVFW06 to handle the *avShowWindow* method.

If a filter or command handler is either started, or already active inside window EMP_WINDOW, then if it has the special 'wake up' method in it, the method will be invoked every time some other window executes:

Mthroutine uShowWindowCompleted Options(*Redefine) #Empno := #Com_Owner.avCurrentWindow.OpenInfo<1> Use message_box_show (ok ok info *Component #Empno) Endroutine

this method will be invoked every time some other window executes:

#AvFrameworkManager.avShowWindow WindowName(EMP_WINDOW)

In other words, this is a method that is saying 'wake up, another window wants you to do something'. Typically the command handler or filter that is 'woken up' would use information passed in the open information strings to determine what it should do next.

- 1. Open the Employee Details command handler (iiiVFW06) in the editor.
- 2. Add the following logic to create the *uShowWindowCompleted* method routine:

```
Mthroutine Name(uShowWindowCompleted) Options(*Redefine)
#EMPNO := #Com_Owner.avCurrentWindow.OpenInfo<1>
Fetch Fields(#XG_HEAD) From_File(PSLMST) With_Key(#EMPNO) Val_E
If_Status Is_Not(*okay)
Message Msgtxt('Employee Details not found for : ' + #EMPNO)
Endif
#fullname := #SURNAME + ', ' + #GIVENAME
#com_ancestor.avSignalEvent Withid(Add_List_Entry) Sendainfo1(#EMPNO)
#DEPSEC_DD.ucurrDept := #deptment
#DEPSEC_DD.uCurrsection := #section
Endroutine
```

## Note:

• DEPSEC_DD is the department / section dropdown component
(iiiVFW14) which was added in exercise VFW074.

- 3. Review the code in Step 2 and note that:
  - Employee number is retrieved using #Com_owner.uCurrentWindow.OpenInfo<1> which was set up by the avShowWindow in the *Employee Images for Section* command handler.
  - Required fields are retrieved from the employee file.
  - An avSignalEvent is used, to tell the active filter to add an entry to the instance list.
- 4. Compile the changed Employee details command handler IIIVFW06.
- 5. Execute your framework and display the *Images* command handler for a suitable Department / Section (in which one or more employees have images recorded).
- 6. Display the pop-up menu for an employee image and select the Open Employee Details Window option. A second framework window should be opened.

Notice that employee details are not displayed for the first employee requested.

7. Select another employee image and select *Open Employee Details Window* again from the conext menu. This time, notice that in the new window an entry is added to the instance list and the correct employee details are displayed.

Do you know why this is happening?

## **Step 4. Ensure Details displayed for first Employee**

As explained in Step 3. Change Employee Details Command Handler, the second window must be open in order to process the *uShowWindowCompleted* method. This means your initial *avShowWindow* in *Employee Images for Section* command handler will not be processed, except to open the new window III_EMP.

Bear in mind that this is simply an exercise that illustrates how to programmatically show a command in a new window. However, as mentioned earlier, a user could simple use the framework's *Windows* menu to open the Employees business object in a new window, and manually access the required employee details.

Employees (Main Window) Ouick Find ... Training - 2JUL12 Open a New Window .. 0 Close All Windows Administration Queues Overall theme Favorites HR Demo Application 63 iii HR Application **Open Application iii HR Application** By Name 😥 By Location 63 JI HR Application 盒 Departments Clear List Marcus Instance Example Employees Sean 💕 23 **Programming Techniques** Reports O. nployee Surname Sections A1002 Smythe John Smithe Robert ELT.

Si

35,

34

For example:

The following solution uses a timer component to invoke an *avShowWindow* to open a new framework window and then, after a short delay, invoke *avShowWindow* again so that the details for the first employee selection are displayed.

1. With Employee Images for a Section open in the editor, on the *Design* tab, drag a timer component onto the panel. The *Timer* is a non-visual component and will not be displayed in the *Design* view.

Define_Com Class(#PRIM_TIMR) Name(#TIMR_1)

2. Select the above line and press F7 to display the *Details* tab for the Timer. Change its *Interval* property to **0**. The timer will initially not be started.

Define_Com Class(#PRIM_TIMR) Name(#TIMR_1) Interval(0)

3. Create a NewWin method routine:

```
a.Move all the code from the uOpenEmpWindow event routine to the NewWin method routine.
```

b.In the NewWin method routine delete this line:

#empno := #i_empno

Your **NewWin** routine should now look like the following:

```
Mthroutine Name(NewWin)
* define a temporary class #VF SY154 reference
Define_Com Class(#vf_sy154) Name(#window) Reference(*dynamic)
* Ask the framework manager to locate a window by name and return a referer
#window <= #avFRAMEWORKMANAGER.avWindow<'III EMP'>
* True if second window is open
If (#Window *IsNot *Null)
Message Msgtxt('Window III_EMP is already open')
* Display employee details in III_EMP window
#AvFrameworkManager.avShowWindow Caller(#Com Owner) For(BUSINES
* Second window needs to be opened and called again after a short delay
Else
* Open a new VLF window and start timer
#AvFrameworkManager.avShowWindow Caller(#Com Owner) For(BUSINES
Endif
* Free VF SY154 reference
#window <= *null</pre>
Endroutine
```

4. Change the NewWin method routine:

a.Add *OpenInfo1(OPEN*) to the second avShowWindow line.

b.In the *Else* logic, set the Timer Interval to 2000. This will start the timer with a tick event every 2 seconds.

Your new code should look like the following. New code is highlighted:

```
Else

* Open a new VLF window and start timer

#AvFrameworkManager.avShowWindow Caller(#Com_Owner) For(BUSINES

OpenInfo1(OPEN)

#TIMR_1.Interval := 2000
```

Endif

5.Save your changes.

6.Complete the uOpenEmpWindow event routine:

a.Change Empno to i_Empno

b.Invoke the NewWin routine

Your code should look like the following:

Evtroutine Handling(#ImageCollection<>.uOpenempwindow) Uempnum(#i_empno)

#empno := #i_empno

#### * Open employee details in a new framework window

#com_owner.NewWin

Endroutine

#### 5. Review the **NewWin** method routine logic.

- Create a temporary reference to VF_SY154, named Windows
- Check if window III_EMP exists and then:
  - Message "Window is already open"
  - Invoke avShowWindow for business object EMPLOYEES, command handler DETAILS, in window III_EMPNO and passing EMPNO as *OpenInfo1()*
- Else
  - Invoke avShowWindow for business object EMPLOYEES, command handler DETAILS in window III_EMP and pass OPEN as *OpenInfo1()*.
  - Message "Window will open shortly"
  - Set Timer Interval property to 2000.
- Set Window to null
- 7. Add the following event handling routine for the *Timer Tick*:

```
Evtroutine Handling(#timr_1.tick)
#std_count += 1
If (#std_count = 2)
#com_owner.NewWin
#timr_1.stop
Endif
Endroutine
```

The Timer is set to an tick interval of 2 seconds (*Interval(2000)*). This routine will give a delay of 4 seconds , which on a modern PC will ensure the second window is running before a second request is made to show the selected employee. You may need to adjust the *TIMR_1.Tick* event to allow for a slower PC.

- 8. Compile the *Employee Images for a Section* command handler.
- 9. Execute your framework. Select a *Department / Section* with employees having images and use the pop-up menu *Open Employee Details Window*. The new framework window will be displayed and after a short delay, the correct details for the selected employee should be displayed.

### Summary

## **Important Observations**

- *avShowWindow* opens a new window or interacts with an already open second window.
- *uShowWindowCompleted* enables a command handler to perform any processing required when *avShowWindow* is used in another window and command handler.

## **Tips & Techniques**

• An alternative simple solution for this requirement could be a push button on the *Employee Images for Section* command handler, which opens a new window and enables a pop-up menu item. When the *Open Employee Details Window* menu option is then used, the second window will already be open. A timer would not be required.

## What I Should Know

• How to handle opening a new window and interacting with it.\

## VFW124 – Using Business Object SubTypes

## What are SubTypes?

Business objects may optionally have a *SubType* associated with them.

For example, a business object named BankAccount might be sub-typed as being a Savings, Check (Cheque) or Investment Account.

The purpose of subtypes is to allow the display of commands (and their associated tabs) for the business object to be conditioned so that they are only visible and useable for certain subtypes.

For example, the command/tab named Transactions might only be validly displayed for Check and Investment accounts.

Likewise, the command/tab named Charges might only make sense when used with a Savings account.

Subtypes are represented by a code that you can associate with a business object instance. For example you might choose the codes SAV, CHK and INV for the three BankAccount subtypes.

You specify how commands and subtypes are related by using the *SubTypes* tab associated with the business object. For example:

ed for SubType(s) Default Yes RS Never RS Never RS Never No RS Never
Yes Never 25 Never 25 Never No 25 Never No 25 Never
25 Never 25 Never 25 Never No 25 Never
25 Never 25 Never No 25 Never
RS Never No No Never
No Never
RS Never
RS Never

Subtypes should be no more than 5 characters long, and contain uppercase letters of the English alphabet (A–Z) or numbers (0–9) only.

The values ALL, NONE, ALLOTHERS and OTHERS should not be used for SubTypes.

SubTypes are only applicable to instance level commands/tabs.

Any Command that is not enabled for all subtypes needs to have its Default Command option set to **Never** 

Once you start using subtypes for a business object instance list you should use them for every instance list entry.

When you insert or update an entry into an instance list you may optionally specify a *SubType* to be associated with the entry.

## Objective

- To create an Accounting application with an Account business object using the *Instant Prototyping Assistant*.
- Account actions to include Details, Approve, Dates, History, Transactions, Charges and Schedule.
- To define Subtypes Check, Savings and Investment.
- To populate the instance list using a hidden filter.
- To implement Account actions using a common demonstration command handler.

To achieve these objectives you will complete the following:

Step 1. Create a Prototype Accounting Application

Step 2. Define Accounts Subtypes

Step 3. Create a Hidden Filter for iii_Accounts

Step 4. Create a Dummy Accounts Command Handler Summary

## **Before You Begin**

Complete exercises VFW030, VFW040 and VFW042.

### **Step 1. Create a Prototype Accounting Application**

- 1. Start the Framework as a Designer.
- 2. Start the Instant Prototyping Assistant from the Framework menu.
- 3. Define an Accounts business object.
- 4. Define additional actions: Approve, Dates, History, Transactions, Charges and Schedule
- 5. Associate the actions Details, Approve, Dates, History, Transactions, Charges and Schedule with Accounts.
- 6. Define an iii_Accounting application and associate the Accounts business object with it.
- 7. Click Finish to create your prototype.
- 8. Save and restart your Framework.



## **Step 2. Define Accounts Subtypes**

- 1. Open the *Properties* dialog for business object Accounts.
- 2. Select the Instance List / Relationships tab.
- 3. Set up the instance list columns as per the table:

Seq	Туре	Caption	Width	Decimals	Edit Code
10	VisualID1	Account Number	15		
20	VisualID2	Account Name	30		
30	AColumn1	Account Type	15		
40	NColumn1	Account Balance	40	2	2

- 4. *Save and Restart* your Framework. Select Accounts business object and fill the instance list using the *Emulate Search* button, to confirm your instance list definition
- 5. Account Subtypes will be defined based on the following:

Account Type	Subtype
CHECK	CHK
SAVINGS	SAV
INVESTMENT	INV

6. Open the Properties dialog for Accounts and select the Subtypes tab.

Identification Icons	Visual Styles   Filters   Filter Set	tings Commands Enabled (	Comn 1
Commands	Enabled for SubType(s)	Disabled for SubType(s)	Default
Approve	ALL	NONE	Yes
🛗 Dates	ALL	NONE	No
History	ALL	NONE	No
Transactions	ALL	NONE	No
😽 Charges	ALL	NONE	No
Schedule	ALL	NONE	No
Detais	ALL	NONE	No
Approve Enter enabled / disabled Enabled for SubTrac(s)	business object subtypes as cor	mma separated values	
<ul> <li>Approve</li> <li>Enter enabled / disabled</li> <li>Enabled for SubType(s)</li> </ul>	business object subtypes as cor	nma separated values	
Approve Enter enabled / disabled Enabled for SubType(s) Disabled for SubType(s)	d business object subtypes as cor ALL NONE	nma separated values	
Approve Enter enabled / disabled Enabled for SubType(s) Disabled for SubType(s)	d business object subtypes as con ALL NONE	mma separated values	

7. Select each Command and enable and disable it for Subtypes based on the following table:

Command	Enable for Subtypes	Disable for Subtypes
Details	ALL	NONE
Approve	СНК	OTHERS
Dates	СНК	OTHERS
History	ALL	NONE
Transactions	CHK, INV	OTHERS
Charges	SAV	OTHERS
Schedules	INV	OTHERS

8. *Save and Restart* your Framework. In the next step you will create a hidden filter to populate the instance list, which will enable you to see your Subtypes working.

## Step 3. Create a Hidden Filter for iii_Accounts

You are developing a demonstration application which runs without real data or files. The hidden filter will populate the instance list with a fixed set of accounts values.

1. Create a Reusable Part / Object:

#### Name: iiiVFW39

#### Description: Accounts Hidden Filter

2. Replace the components code with the following:

```
Begin_Com Role(*EXTENDS #VF_AC007) Height(148) Hint(*MTXTDF_FI
                    ______
* Method Definitions
* _____
* Handle initialization of this filter by redefining the default behaviour
Mthroutine Name(uInitialize) Options(*Redefine)
* Invoke the uInitialize behaviour in the ancestor
Invoke Method(#Com_Ancestor.uInitialize)
* Make this a hidden filter
Set Com(#Com_Owner) Avhiddenfilter(TRUE)
* Make up a set of dummy bank accounts ....
* Indicate start of list update and set the framework to busy
Invoke Method(#avListManager.BeginListUpdate)
* Clear the list first
Invoke Method(#avListManager.ClearList)
* Make up some dummy accounts
Execute Subroutine(Add) With Parms('67383940' 'Fred Bloggs' 465.12 SAVIN
Execute Subroutine(Add) With Parms('73839915' 'Fred Bloggs' 34567.78 INV
Execute Subroutine(Add) With Parms('74849192' 'Fred Bloggs' 12354.56 CHI
Execute Subroutine(Add) With Parms('74848949' 'Mary Smith' 1465.12 SAVI
Execute Subroutine(Add) With Parms('51617283' 'Mary Smith' 354.56 CHEC
Execute Subroutine(Add) With Parms('71828234' 'Mary Smith' 347.78 INVES
Execute Subroutine(Add) With_Parms('91828373' 'Mary Jones' 5162.45 SAVII
Execute Subroutine(Add) With_Parms('71726364' 'Mary Jones' 167.89 CHECI
Execute Subroutine(Add) With_Parms('84849596' 'Mary Jones' 65363.67 INV.
* Indicate end of list update
Invoke Method(#avListManager.EndListUpdate)
Endroutine
```

* Subroutines * _____ * Add accounts to the list with appropriate subtypes Subroutine Name(ADD) Parms((#T_ACC *Received) (#T_NAM *Received) ( Define Field(#T_ACC) Type(*char) Length(10) Desc('Account Number') Define Field(#T_NAM) Type(*char) Length(50) Desc('Account Description') Define Field(#T_BAL) Type(*Dec) Length(11) Decimals(2) Desc('Account Ba Define Field(#T_TYP) Type(*char) Length(20) Desc('Account Type') Case Of_Field(#T_TYP) When Value_Is('= SAVINGS') Invoke Method(#avListManager.AddtoList) Akey1(#T_ACC) Visualid1(#T_A When Value_Is('= CHECK') Invoke Method(#avListManager.AddtoList) Akey1(#T_ACC) Visualid1(#T_A When Value Is('= INVESTMENT') Invoke Method(#avListManager.AddtoList) Akey1(#T_ACC) Visualid1(#T_A Endcase Endroutine End Com

* ______

- 3. Compile the reusable part and review the logic.
  - a. The subroutine *ADD* is passed parameters of account number, account name, account balance and subtype and adds an instance list entry with appropriate values.
  - b. Note that AColumn1 contains the visible account type value and Subtype contains the subtype code.
  - c. The *uInitialize* method execute the *ADD* subroutine nine times to set up the instance list.
- 4. Open the *Properties* dialog for the Accounts business object. Select the *Filters / Snap In filter settings* tab and plug in the hidden filter, iiiVFW39. Use the *Find* dialog, which will plug in using Identifier.
- 5. *Save and Restart* your framework. Select Accounts and your filter should now be populated:

VBBE	1666		
Account Number	Account Name	Account Type	Account Balance
51617283	Mary Smith	CHECK	354.56
67383940	Fred Bloggs	SAVINGS	465.12
71726364	Mary Jones	CHECK	167.89
71828234	Mary Smith	INVESTMENT	347.78
73839915	Fred Bloggs	INVESTMENT	34567.78
74848949	Mary Smith	SAVINGS	1465.12
74849192	Fred Bloggs	CHECK	12354.56
84849596	Mary Jones	INVESTMENT	65363.67
91828373	Mary Jones	SAVINGS	5162.45

## **Step 4. Create a Dummy Accounts Command Handler**

Once again, as you are developing a demonstration application, the command handler will be common for all Accounts actions.

1. Create a new *Reusable Part / Panel*:

#### Name: iiiVFW40

#### Description: Account Command Handler

2. Replace its code with the following:

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #VF_AC010) Height(242) Width(728) Define_Com Class(#STD_TEXTL.Visual) Name(#STD_TEXTL) Caption('Acc Define_Com Class(#STD_OBJ.Visual) Name(#STD_OBJ) Caption('Account I Define_Com Class(#STD_TEXTS.Visual) Name(#STD_TEXTS) Caption('Bu Define_Com Class(#PRIM_LABL) Name(#LABL_1) Caption('This example u Define_Com Class(#PRIM_LABL) Name(#LABL_2) Caption('SAV (Saving) Define_Com Class(#PRIM_LABL) Name(#LABL_3) Caption('CHK (Check) Define_Com Class(#PRIM_LABL) Name(#LABL_3) Caption('INV (Investme Mthroutine Name(uExecute) Options(*Redefine) Invoke Method(#Com_Ancestor.uInitialize) Invoke Method(#avListManager.GetCurrentInstance) Akey1(#STD_OBJ) Visu Endroutine End_Com

- 3. Compile the reusable part and review its logic.
  - a. The user interface contains simply account number, name and subtype.
  - b. The *uExecute* method, retrieves these three values for the current instance list entry.
- 4. Open the *Properties* dialog for Accounts. Select the *Commands Enabled* tab.
- 5. Select each command and plug in the command handler, iiiVFW40.

Plug in the **Approve** command handler using the *Find* dialog, so the plug in uses the *Identifier*. Note the *Identifier* name and plug this in to all other commands for **Accounts**.

- 6. *Save and Restart* your Framework.
- 7. Test your Accounting application which demonstrates the use of Subtypes.

Select instance list entries for each subtype and confirm the correct commands are enabled.

### Summary

### **Important Observations**

• Subtypes are optional.

### **Tips and Techniques**

• Subtypes provide an easy way to control the commands which are enabled for different types of the same object.

## What You Should Know

• How to implement Subtypes in your own applications.

## VFW126 – Using Space Objects (Optional)

## **Introduction to Spaces**

Space objects are primarily designed to support 'batch' style functions that process large volumes of information. You can load the required information into a space at the start of a function and then repeatedly (re)access the indexed space more efficiently than you can access the DBMS directly.

Space objects are unique within a process by their name. Space objects cannot be shared between processes. Space objects are not persistent. A Space object and its data content cease to exist when the process that owns them ends.

Space objects are defined and accessed using Built In Functions (BIF). A BIF is a call to program which has a defined set of inputs and outputs. BIFs are defined in the *Repository* and this means that the editor supports BIFs with a predefined set of input and output parameters.

BIFs are accessed using the USE command.

## Objectives

A *Salary* command handler for the Reports business object will display a list of all employees. The *Salary* command handler also defines a space object. All employees are added to the space object, keyed by their department code and employee number.

A second command handler, *Employee Query*, will read entries in the space object by department code and display a list of currently selected employees in this department.

The space object exists within a single Windows process. For a VL Frameworks application, this means all components running within a Framework could access the space object, once it has been created.

Reports	_					
Favorites     & HR Demo Application     # # HR Application	C Report	: Salary ee Query 🏾 🍣 Salar	y 📄 Weekly			(
Departments	Employ	Employee Surname	Employee Given N	Employee Sa	Selected	A
Employees	A0070	BROWN	VERONICA	50,125.00		
Reports	A0090	BLOGGS	FRED JOHN ALAN	20,045.91		
iii Transport Application	A0193	SIMPSON	FRED	35,000.04		
The second second	A0907	JONES	ANNE	34,213.04		
I I IR Application	A1001	JONES	BEN	2,345.82		
Marcus Instance Example	A1002	SMYTHE	JOHN	25,000.04		
Programming Techniques	A1003	SMITHE	Robert	31,000.04	YES	
Administration	A1004	SMITHSON	PAUL	21,000.04		
	A1005	SMITHS	PETER	46,700.04		
	A1006	SMITHERS	JACK	25,000.04		
	A1007	SNELL	GEORGE	26,780.04		
	A1008	SNEDDON	ALLAN	450,000.04		
	A1009	SNASHALL	DAMIAN	31,000.04	YES	
	A1010	PERRY	WILLIAM	60,000.04		
	A1011	PERRIN	CHRISTOPHER	25,000.04		
	A1012	PAUL	PATRICK	26,456.04		
	A1013	PATTISON	GEORGE	78,977.04	YES	
	A1014	MOORE	JOHN	68,000.04		
	A1015	WOODS	BRADLEY	313,000.04		
	A1016	TURNER	JACK	22,000.04	(1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	
	A1017	NEAVE	GARY	25,600.04	YES	
	A1018	ZACHARIA	PAUL	25,900.04		
	A1019	DICKENS	CHARLES	45,000.04		
	A1020	DOUGLAS	ADAM PETER	121,500.04		1
	•					•
			Total S	Salary	187,577.2	20
a 📾 🗃		Messages	Ready Local	ENG JI	VORY12 8/08/	/12 16:04

To meet these objectives you will complete the following steps:

Step 1. Create the Salary Command Handler

Step 2. Create the Employee Query Command Handler.

Step 3. Add Logic to the Employee Query Command Handler Summary

### **Before You Begin**

Complete exercises VFW030, VFW040 and VFW042.

## Step 1. Create the Salary Command Handler

1. Create a new Reusable Part / Panel:

#### Name: iiiVFW42

#### Description: Salary Command Handler for Reports

- 2. Give the reusable part an ancestor of VF_AC010
- 3. Use the *Design* ribbon to give iiiVFW40 an *Attachment manager*
- 4. Drop a Panel component onto the bottom of the main panel. Change the *Name* to **BOTTOM_PANEL**.
- 5. Drop a List View into the center of the main panel, so that it occupies the rest of the space.
- 6. If necessary use the *Layout Helper* tab to confirm the position of the child panel and list view.
- 7. Save the design.
- 8. If field TOTSALARY does not already exist, create a new Packed Decimal field, with a length of 15 and 2 decimal places. Give it an Edit Code of 2. Drag and drop field TOTSALARY onto the right hand side of BOTTOM_PANEL. Resize the field as necessary.
- 9. Locate the file PSLMST in the Repository and drag and drop fields EMPNO, SURNAME, GIVENAME and SALARY into the List View. Finally drop field STD_TEXTS onto the list view.
- 10. Change the *Caption* for the STD_TEXTS column to **Selected** and change *CaptionType* to **Caption**.
- 11. Create an *Initialize* event for the list view.
- 12. Save the design.
- 13. Switch to the *Source* tab.
- 14. Add the following code to the list view *Initialize* event handling routine to define the space object:

#std_text := IIIVFW42

* Create Space with the name entered in std_texts field

Use Builtin(CREATE_SPACE) With_Args(#std_text) To_Get(#STD_CMPAR) * Define Space Cells

Use Builtin(DEFINE_SPACE_CELL) With_Args(#STD_TEXT DEPTMENT

Use Builtin(DEFINE_SPACE_CELL) With_Args(#STD_TEXT EMPNO KEY Use Builtin(DEFINE_SPACE_CELL) With_Args(#STD_TEXT GIVENAME) Use Builtin(DEFINE_SPACE_CELL) With_Args(#STD_TEXT SURNAME) Use Builtin(DEFINE_SPACE_CELL) With_Args(#STD_TEXT SECTION) Use Builtin(DEFINE_SPACE_CELL) With_Args(#STD_TEXT SALARY)

The code above has created a space object with same name as this component. The name can be any alphanumeric value, but must be unique within the Windows process (that is, within the job).

**Hint:** Type Use Builtin(DEFINE_SPACE_CELL) and then press *F4* to display the *Command Assistant*.

Expand the WITH_ARGS parameter and note that the editor shows the details for the parameters required.

That is, Space Object Name and so on.

Assistant	Succession Succession	a minute state
💢 🧹 Use Builtin(DEFINE_SPACE_CELL) With	Args(#STD_TEXT DEPTMEN	T KEY)
BUILTIN	DEFINE_SPACE_CELL	$\checkmark$
- WITH_ARGS	#STD_TEXT DEPTMENT	Commands Expression Var
Space Object Name	#STD_TEXT	
Name of prototype field (without # sign)	DEPTMENT	Resulting expression
Cell attributes (eg KEY DESCEND NOCASE)	KEY	#STD_TEXT
- TO_GET		
Return Code (OK or ER)		

15. Define a Group_By for all the employee fields required. Different fields are required for the list view and the space object.

Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME #D

16. Complete the *LTVW_1.Initialize* event routine by populating the list view and inserting each employee into the space object:

```
#std_texts := *blanks
Clr_List Named(#LTVW_1)
#std_count := *zeroes
Select Fields(#empdata) From_File(pslmst)
Add_Entry To_List(#LTVW_1)
* Add to Space object
Use Builtin(INSERT_IN_SPACE) With_Args(#STD_TEXT #DEPTMENT #E
```

```
#std_count += 1
Endselect
Message Msgtxt('Space object IIIVFW42 created with ' + #std_count.asstring -
```

17. The command handler should display the Total Salary for currently selected entries in the list view.

Create *ItemGotSelection* and *ItemLostSelection* event routines for the list view and add appropriate logic in each to maintain TOTSALARY.

The STD_TEXTS column should contain YES when the row is selected.

Your code should look like the following:

Evtroutine Handling(#LTVW_1.ItemGotSelection) Options(*NOCLEARMES #std_texts := 'YES' Upd_Entry In_List(#LTVW_1) #TOTSALARY := #TOTSALARY + #salary Endroutine Evtroutine Handling(#LTVW_1.ItemLostSelection) Options(*NOCLEARMES #std_texts := *blanks Upd_Entry In_List(#LTVW_1) #TOTSALARY := #TOTSALARY - #salary Endroutine

18. Compile the new command handler.

19. Execute the Framework as Designer.

- a. Open the *Properties* dialog for the *Reports* business object.
- b. Select the *Commands Enabled* tab, select the Salary action and plug in the Salary command handler, iiiVFW42.
- c. Use the *Find* dialog, so that the plug in uses the Identifier.
- 20. Save and Restart the Framework.
- 21. Select the Reports business object. Right click on the Weekly command handler and select Salary from the context menu.
- 22. Test the Salary command handler by selecting a number of employees, while holding down the Control key. Note Total Salary shows the total for selected employees.
- 23. Change the selected employees and ensure that the Total Salary is

recalculated.

## Step 2. Create the Employee Query Command Handler.

1. Create a new *Reusable Part / Panel*:

#### Name: iiiVFW43

#### Description: Reports Employee Query Command Handler

- 2. Give the reusable part an ancestor of VF_AC010
- 3. This command handler will display a list of employees from the space object selecting entries using the department code key.

Total Salary is shown for selected employees.

ШV	FW46 - Sele	ct from Space	2				<b>×</b>
De	epartment Co	de ABCD	Se	lect			
En	mployee N	Employee Surr	name	Employee Given N	Department	Section Code	Employee !
VE	ABCDE	ABCDEFGHIJK	1_MNO	ABCDEFGHIJKLM	ABCD	AB	123,456,7
1				III			•
				Total Salary	[	1,234,567,890,1	123.12

Note that the Employee Query displays all fields from the space object, which is different to the list view in the Salary command handler.

- 4. Use the *Design* ribbon to give iiiVFW43 an *Attachment manager*.
- 5. Drop a Panel component at the bottom of the RP's panel and change the new panel's *Name* to **Bottom**. Adjust its height as required.
- 6. Drop a Panel onto the top of the RP's panel and change the new panel's *Name* to **Top**. Adjust its height if required.
- 7. Drop a list view into the center of the RP's panel. If necessary use the *Layout Helper* tab to ensure that the child components are correctly attached.
- 8. Locate the file PSLMST in the Repository and drop fields EMPNO,

SURNAME, GIVENAME, DEPTMENT, SECTION and SALARY into the list.

- 9. Create an *ItemGotSelection* and *ItemLostSelection* event routine for the list view.
- 10. Drop field TOTSALARY onto the right hand side of the panel Bottom.
- 11. Select the panel Top and use the *Design* ribbon to give it a *Flow Across manager*. Open the *Layout Helper* tab and note that the default setting for *Flow Rules* is **LeftToRight**, which is what is required.
- 12. Select the Top panel on the *Design* view. On the *Layout Helper*, select the *Category Margins* on the *Layout Manager Details* tab. Use the **All** setting to give children on this panel a margin of **6 pixels** on all four sides.
- 13. Drop the field DEPTMENT onto the panel Top. Change its *Name* to **DEPT_IN**
- 14. Drop a Push Button component onto the Top panel. Give it a *Caption* of **Select** and change its *Name* to **PHBN_SELECT**.
- 15. Create a *Click* event for the push button.
- 16. Save the reusable part.

## Step 3. Add Logic to the Employee Query Command Handler

- 1. Select the *Source* tab.
- 2. Complete the *PHBN_SELECT.Click* event routine to:

Initialize work fields and use the SPACE_OPERATION BIF and CHECKEXISTENCE to ensure the space object exists.

Add an If / Else / Endif for STD_CMPAR = OK

Issue an error message if the space object is not found.

Evtroutine Handling(#PHBN_SELECT.Click) #std_text := IIIVFW41 #TOTSALARY := *zeroes * Check Space exists Use Builtin(SPACE_OPERATION) With_Args(#std_text CHECKEXISTENC: If (#std_cmpar = OK) * Populate list view form space object Else Message Msgtxt('Space object IIIVFW42 does not exist') Endif Endroutine

- 3. To populate the list view, add the logic:
- Clear the list view
- Use SELECT_IN_SPACE to retrieve entries using DEPT_IN as key.
- Return the SELECT_IN_SPACE status into field STD_CMPAR.
- While STD_CMPAR is equal to OK
- Add an entry to the list view
- Retrieving all space object entries for the department code = DEPT_IN, must be achieved by then using the SELECT_NEXT_IN_SPACE. Note this also returns status as STD_CMPAR.

Your code should now look like the following. New code is highlighted in red.

Evtroutine Handling(#PHBN_SELECT.Click) #std_text := IIIVF31 #TOTSALARY := *zeroes * Check Space exists Use Builtin(SPACE_OPERATION) With_Args(#std_text CHECKEXISTENC) If (#std_cmpar = OK)

Clr_List Named(#LTVW_1)

Use Builtin(SELECT_IN_SPACE) With_Args(#STD_TEXT #DEPT_IN) Dowhile ('#STD_CMPAR *EQ OK')

Add_Entry To_List(#LTVW_1)

### Use Builtin(SELECTNEXT_IN_SPACE) With_Args(#STD_TEXT #DEP] Endwhile

Else

Message Msgtxt('Space object IIIVFW31 does not exist') Endif Endroutine

4. Complete the Reports Employee Query command handler logic by making the List View *ItemGotSelection* and *ItemLostSelect* event routines maintain Total Salary.

Your code should look like the following:

```
Evtroutine Handling(#LTVW_1.ItemGotSelection) Options(*NOCLEARMES
#TOTSALARY := #TOTSALARY + #salary
Endroutine
Evtroutine Handling(#LTVW_1.ItemLostSelection) Options(*NOCLEARMES
#TOTSALARY := #TOTSALARY - #salary
Endroutine
```

- 5. Compile the Reports Employee Query command handler.
- 6. Execute the Framework as Designer.
- 7. Open the *Properties* dialog for the Reports business object.
- 8. Select the *Commands Enabled* tab, select the Employee Query action and plug in the command handler iiiVFW43. Use the *Find* dialog so that the plug in uses the Identifier name.
- 9. *Save and Restart* the Framework.
- 10. Select the *Reports* business object. Right click on the *Weekly* tab and run the *Salary* command handler.
- 11. Again, use the right mouse menu to run the *Reports, Employee Query* command handler and display employees from the space object for department ADM.

- 12. Restart the Framework and this time run the *Reports / Employee Query* command handler immediately. The "Space object does not exist" error message should be shown in the Frameworks status bar.
- 13. Based on what you've learnt in an earlier exercise, you should appreciate that if required you could add framework switch logic to Employee Query, to run the Salary command handler automatically, when the space object does not exist.

### Summary

### **Important Observations**

• While using INSERT_IN_SPACE or SELECT_IN_SPACE Built-In Functions, the field values must be specified in the same sequence as the cells in the space were defined.

## **Tips & Techniques**

• In high volume repeated commands avoid using visually defined fields as mapping values unless absolutely necessary. When a field has been visually defined, mapping into or out of its value is significantly slower because of the underlying visual context.

## What I Should Know

• How to use SPACE Built-In Functions to store and retrieve static data in PC memory.

# VFW130 – Set up the VL Framework for Client/Server Operation

## Objectives

• To demonstrate how to enable a framework for client/server operation.

To achieve this objective you will complete the following:

Step 1. Enable Framework for Client/Server

- Step 2. Check In Files to the Server
- Step 3. Test the Framework in Client/Server mode

## **Before You Begin**

This exercise can only be completed if you have access to an IBM i or Windows server based LANSA system.

This exercise uses an IBM i (iSeries) server.

The LANSA for iSeries partition must have the Personnel System Files (save file PERSYS) imported using the *Administration / Initialize Partition* option. This will import files such as DEPTAB, SECTAB, SKLTAB, PSLMST and PSLSKL which are referenced by a number of these exercises. The files all contain sample data.

Complete exercises VFW030, VFW040, VFW042 and VFW054. In VFW054 you created two files that are used in Step 2: iiiEmpNotes – Employee Notes and iiiEmpImages – Employee Images.

## Step 1. Enable Framework for Client/Server

This step concerns changing settings at framework level.

1. Start the Framework as Designer. Open the *Framework Properties* dialog:



- 2. Select the *User Administration Settings* tab. In the *Sign on Settings* group box, in the *End Users must Signon to this Framework* dropdown list, select **in MS-Windows applications only.**
- 3. Select the radio button, *Users Sign on to a Remote Server to Use the Framework*.



- 4. Save and Restart the VL Framework.
- 5. On the *Log On* form, click the *Work Offline* button:

Log On	COMPANY TITLE information go to impany site>	
User Password	JIVORY13	Log On Cancel Work Offline

- 6. When the framework has started, select *Servers* from the *Administration* menu.
- 7. Click on the *New* button to define a server:

Servers		- 0
Sequence Server	Server Details Identification Icon	ns
1 New Server System	Caption	New Server System 0
	Hint:	0
	Sequence:	1
	Internal Identifier:	86C64585FBF14248A821C5DE82710323
	Unique Identifier:	145
	User Object Name / Type	86C64585F8F14248A821C5DE82710323
	Restricted Access	
	Last Changed	20130114-163441-JIVORY13
New Delete		
$\smile$		LANSA Comms. Admin Close

- 8. On the *Identification* tab, give the server definition any suitable *Caption* such as **Test Server**.
- 9. Select the *Icon* tab and give the server definition any suitable icon.
- 10. Select the *Server Details* tab.

Click on the *LANSA Comms. Admin* button to find the *Partner LU* to use for the IBM i server.

If necessary, first define the server in the LANSA Communications Administrator.

To do this you must know the server name or IP address and the port number used by the LANSA Listener.

Table Host Ro	outes Advanced	l Help						
munications Rout	te Table and Path:							
Program Files (x86	5) LANSA (Connect)	ROUTE.DA	т					
at Day to a								
ost Routes								
Destructure	O attend to the	a seator of	14- d-d-	<b>T</b>	C	Death	1000	_
Partner LU	Qualified Name	Method	Module	Trac	Conn	Pack	IPCO	
Partner LU *LOCAL	Qualified Name	Method Socket	Module	Trac	Conn	Pack	IPCO	
Partner LU *LOCAL COMPOSER01	Qualified Name localhost EARTH	Method Socket Socket	Module	Trac 0 0	Conn 4541 4544	Pack 0 0	IPCO	
Partner LU SLOCAL COMPOSER01 COMPOSER35	Qualified Name localhost EARTH EARTH	Method Socket Socket Socket	Module	Trac 0 0 0	Conn 4541 4544 4000	Pack 0 0 0	IPCO 1 1 1	
Partner LU PLOCAL COMPOSER01 COMPOSER35 EARTH	Qualified Name localhost EARTH EARTH EARTH	Method Socket Socket Socket Socket	Module	Trac 0 0 0 0	Conn 4541 4544 4000 4543	Pack 0 0 0 0	IPCO 1 1 1 1	
Partner LU COMPOSER01 COMPOSER35 EARTH EARTH13	Qualified Name localhost EARTH EARTH EARTH EARTH	Method Socket Socket Socket Socket	Module	Trac 0 0 0 0	Conn 4541 4544 4000 4543 4537	Pack 0 0 0 0	IPCO 1 1 1 1	
Partner LU PLOCAL COMPOSER01 COMPOSER35 EARTH EARTH13 EARTH12	Qualified Name localhost EARTH EARTH EARTH EARTH EARTH	Method Socket Socket Socket Socket Socket	Module	Trac 0 0 0 0 0	Conn 4541 4544 4000 4543 4543 4542	Pack 0 0 0 0 0	IPCO 1 1 1 1 1 1	

A routing table entry enables LANSA communications to connect to a LANSA Listener on a server using the required Qualified Name and Port.

11. Set up the server details shown based on your own server. For example:

Server Type	LANSA for System i
Server Name	EARTHD12
Partition	TRN
Partition is enabled for RDMLX	Yes
Upper and Lower Case Password	Yes

Servers			C Región, W
Seq Server	Server Details Identification Icons		
1 Test Server	Server Type:	LANSA for System i	DBCS Capable
	Server Name:	EARTH13	Commitment Control
	Partition	TRN	Divert Locks
	Hent Server Translation Tables	*306	Partition is enabled for RDMLX
	Server-Client Translation Table:	*JOB	Use Windows Credentials
	Selection Block Size:	500	Upper and Lower Case Password
	Selection Limit:	10000	Show on Connect Dialog

If necessary change the settings shown to suit your partition definition and password standards.

- 12. Close the Servers dialog and Save and Restart the Framework.
- 13. In the *Log On* dialog enter your User ID and Password for the IBM i. If you are completing these exercises using a *Slave Workstation* installation, you should use your Visual LANSA User ID and Password.
- 14. When your framework has started, your application will now access all files on the IBM i server.

The local files in the Microsoft SQL Server database will no longer be accessed.

15. Close your framework which will currently produce errors if you access your iii HR Application.

## Step 2. Check In Files to the Server

In exercise VDW054 you created two new files:

iiiEmpNotes – Employee Notes iiiEmpImages – Employee Images

In order to run your iii HR Applications in client/server mode, you must check in and compile your new files. The check in and compile will create physical and logical files and OAM programs.

Since you are using an RDMLX-enabled partition, which is required for these exercises, the compile will produce an RPG OAM program for RDML-enabled functions and components, and a C OAM program for RDMLX-enabled functions and components.

Note that these new tables will not contain data.

1. On the *Repository* tab, select files iiiEmpImages and iiiEmpNotes and use the context menu to select *Check in*:



2. In the *Check in Options* dialog, expand the Files entry, select both files and click the *References* button:



3. Select the new fields (field names beginning iii) and click the *Add for Check in* button:

•)			
Name	Description	Allow check-in	Qualifier
iiiEmpImages	Employee Images	V	D13TRNLIB
BLANKS	Blank / blanks variable	$\checkmark$	
ZERO	Zero (0) variable	$\checkmark$	
🖻 📴 🖉	Field update / access identifier	$\checkmark$	
EMPNO	Employee Number	$\checkmark$	
🖻 📑 iiiempimg	Employee Image	$\checkmark$	
iiiEmpNotes	Employee Notes	$\checkmark$	D13TRNLIB
BLANKS	Blank / blanks variable	$\checkmark$	
E TERO	Zero (0) variable	$\checkmark$	
🖻 📴 @@UPID	Field update / access identifier	$\checkmark$	
EMPNO	Employee Number	$\checkmark$	
iiiLneCnt	Line Count	$\checkmark$	
🖻 📜 iiiNteLne	Note Line	$\checkmark$	
iiiNteSqn	Note Sequence	$\checkmark$	
iiiNteType	Note Type	$\checkmark$	

4. Close the *Local Cross References* dialog. Note that the fields have been added to the check in list of objects:


5. Select the check in options to *Rebuild table*, *Rebuild Indexes* and *Rebuild OAMs* and click OK to start the check in.

**Note:** The *Delete* \$\$ *file* option will not be required as these are new files.

Usually, whenever you check in and compile a file, this option should be checked to delete the copy file created by the previous check in and compile.

6. Check the *Check In* tab to confirm your fields and files were checked in successfully.

×	p .	Job Status	Description	Results	Currently Processing	Started
#	×	Completed	Check in 7 objects	0 fatal errors - 0 warnings	EARTH13 - RDML Compiles	15/01/2013
1		<ul> <li>Objects</li> </ul>				
1	~	📜 iiiempimg	Employee Image			
3	^	🔚 iiiLneCnt	Line Count			
1		🣜 iiiNteLne	Note Line			1
		📑 iiiNteSqn	Note Sequence			
		裙 iiiNteType	Note Type			
		🔁 iiiEmpImages	Employee Images			(
1		🔁 iiiEmpNotes	Employee Notes			
		4				
4.0	amp	ile A Check In A Chec	k Out	n 🔲 Import 🖗 Help		

7. You should now be able to execute your iii HR Application in client/server mode to the IBM i server.

## Step 3. Test the Framework in Client/Server mode

- 1. Execute the Framework as user. Note that the *Log On* dialog requires a log in to the server. As a user you cannot run the Framework offline unless this option is configured by the designer.
- 2. Log in with your IBM i user id and password.
- 3. Select the iii HR Application and work with Departments, Employees, Reports and Sections. All your filters and command handlers should perform as before.

Employee Brief Notes, Employee Notes, Employee Images and Employee Images for a Section will require you to set up new data, as you are now using the files on the IBM i server.

4. Close your Framework.