Visual LANSA Fundamentals Windows Tutorials

The exercises in these tutorials are designed to provide you with an overview of the basic skills for working with the LANSA Editor and the LANSA Repository, as well as providing some practical examples of using the tools LANSA provides to help you code your applications faster using Forms.

The tutorials are intended to be done in this sequence:

- User Interface Tutorials
- Repository Development Tutorials
- LANSA Editor Tutorials
- RDML Programming using Visual LANSA Forms
- An introduction to the VISUAL LANSA Framework

Supporting material:

• Appendix A. Personnel Demonstration System

Edition Date January 6, 2014 © LANSA

About the Tutorials

Following is information for you to prepare for completing the exercises in these LANSA Fundamentals Tutorials:

Tips for using the Exercises

How many Developers can use the Exercises?

What Partition should I use?

Using Long Names

Tutorial Installation

and finally, a link to our support department, should you have any suggestions for improving these exercises: Your Feedback

Tips for using the Exercises

You need to complete these tutorials in sequence because fields and files created in early exercises are required to complete the later exercises.

Exercises are designed to be completed in sequence as later exercises are designed to use skills from the earlier exercises.

- Check off each step in an exercise as you complete it.
- Follow the instructions very carefully.
- The first steps in an exercise will provide very precise descriptions of the tasks to be performed. As the steps and course progresses, the instructions will become much more general.

Before you start, ensure that the environment that you will be using is set up as described in What Partition should I use?

How many Developers can use the Exercises?

There is no limit to the number of developers who may use the training at the same time. However, it is important that each developer has a unique identifier for their own work.

In the exercises, each developer will use an object prefix **iii** which can be based on their initials or could be assigned by a system coordinator.

To allow for more than one developer to use the exercises, prefix any objects you create with a unique identifier such as your initials (**iii**).

Note that if you are using a trial version of Visual LANSA without a license, you cannot use the iii prefix but must use DEM instead. Following are the identifiers you may use:

File: DEMFIL01 to 10 Form / Reusable Part / WAM: DEMCOM01 to 10 Process: DEMPRO01 to 10

Function: DEMFN01 to 10

If you are using a shared server Repository, only one student may complete the course at a time as names must be unique. See also Using Long Names.

What Partition should I use?

The partition you use has to be enabled for Full RDMLX with the Demonstration Material installed.

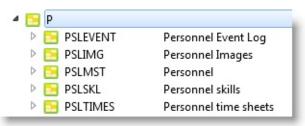
To verify that the partition has been set up for the exercises:

- 1. In the Visual LANSA editor, open your partition definition and display the RDMLX settings.
 - a. Ensure the option Enable Partition for full RDMLX has been selected.
 - b. Ensure the option Enabled for Long Names has been selected.



To verify that the Demonstration Material has been included:

- 1. Expand the Files node in the Repository tab, locate files starting with letter P.
- 2. Make sure that the files PSLEVENT, PSLMST, PSLSKIL and PSLTIMES are present.



If these files are not present, and you are working with an independent LANSA System:

- 1. Close down LANSA Development Environment
- 2. Restart LANSA Development Environment
- 3. Reinitialize the partition using the Partition Init... button in the Visual LANSA Logon screen.

User ID: Password:			
Use Window	vs credentials		
Partition	Current Language	Task ID	
SYS	English	TRAINING	
TRN	French	TSYS0001	

4. In the Partition Initialization dialog select the Personnel System Demonstration material option.

	Partition	Initializatio	n	×
Mandato	ry Partition Initializat	ion		
Visual LA	ANSA Framework			
Enable f	or the Web			
LANSA C	client field and file de	finitions		
Demonst	tration material			
Run Den	nonstration			
ОК	Show Last Log	Messages	Cancel	Help

3. Press OK and wait for the import to finish.

For details, refer to the Partition Initialization options in the *Visual LANSA Administration Guide*.

If you are working with a slave LANSA system and the partition is not set up as required, contact your LANSA server administrator.

Using Long Names

These tutorials assume that you are using Long Names. This means 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).
- 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). Identifier cannot be changed once a new object has been saved.

If you are using a Trial Copy

• If you are using a trial copy of Visual LANSA (that is, no licence) you can create files and forms using a long name, but as you create the object you must assign the restricted object Name as the Indentifier. For example:

🔁 New File	-	Long	g Name	
Name	iiiEmployeesFile		Create	
Description	Employees file		Cancel	
Library	DC@DEMOLIB			
Identifier	DEMFIL03		ntifier Dpen in editor	
Enabled For RDMLX	V	DEIWIFI		

• For more details refer to LANSA Object Names in the *Technical Reference Guide*.

Tutorial Installation

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

Your Feedback

Your feedback regarding these exercises will help us improve the overall quality of the LANSA documentation and training. Please email your comments to LANSA Training support.

User Interface Tutorials

What is the User Interface Tutorial?

This tutorial contains exercises that are designed to introduce and reinforce the fundamental user interface skills required to use the Visual LANSA Development Environment. They focus on the user interface, navigation, and search techniques as opposed to teaching programming skills to build applications. (Refer to the Repository Development and Programming using Visual LANSA Forms Tutorials to learn LANSA development skills.)

While all the essential areas are covered to enable a new LANSA developer to get started. They are not comprehensive. The User Interface tutorials include:

VUI001 – Starting LANSA VUI002 – LANSA Editor Parts VUI003 – Repository Tab VUI004 - Details, Outline, Favorites Tab

Before you Begin

You must have LANSA Demonstration Personnel System installed in the partition that you will use with the set up options as described in What Partition Should I Use?.

The LANSA Demonstration Personnel System contains all the objects used by these exercises.

Tips for using the exercises

- Check off each step in the exercise as you complete it.
- Follow the instructions very carefully.
- Remember to replace iii with your unique 3 characters. You will not always be reminded to make this substitution.

For further information refer to How many developers can use the exercises?

• These exercises assume that you have not previously customized the editor interface. If you have already customized your environment, the example screens and instructions may not exactly match your customized development environment.

The following are important notes regarding the structure of the exercises:

• The first steps in an exercise will provide very precise descriptions of the

tasks to be performed. As the steps and course progresses, the instructions will become much more general.

• Later exercises are designed to use skills from the earlier exercises. These exercises are designed to be completed in sequence.

VUI001 – Starting LANSA

Objective:

- To learn how to start Visual LANSA.
- To learn how to logon using a specific partition, language and task ID.
- To learn how to use the LANSA documentation and toolbar.
- To learn how to use context sensitive help in the user interface.

To achieve these objectives, you will complete the following:

Step 1. Starting the LANSA Development Environment

Step 2. Online Documentation

Step 3. F1 Context Sensitive Help

Summary

Before You Begin:

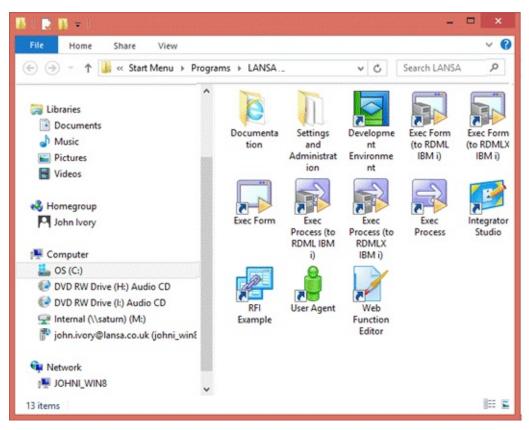
You may wish to review the following topics:

• Visual LANSA Logon in the Visual LANSA Administrator's Guide

Step 1. Starting the LANSA Development Environment

In this step you will locate the LANSA folder and start LANSA.

1. The Visual LANSA Installation creates a LANSA desktop folder a shown:



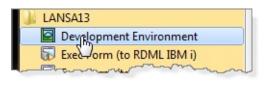
If you open the *Documentation* folder, you can directly access the LANSA online documentation. Help (also referred to as the *online guide*) can also be accessed from within the *Visual LANSA IDE*. The IDE includes extensive context sensitive help links.

If you open the *Setting and Administration* folder, you will see a group of program icons used by LANSA Administrators to configure the LANSA environment:

2. Start LANSA using the *Development Environment* icon 粒 in the LANSA folder

or

via the *Development Environment* icon in the *Windows Start Menu*.





3. Double-click on the Environment icon to start LANSA. The Visual LANSA Logon dialog will open:

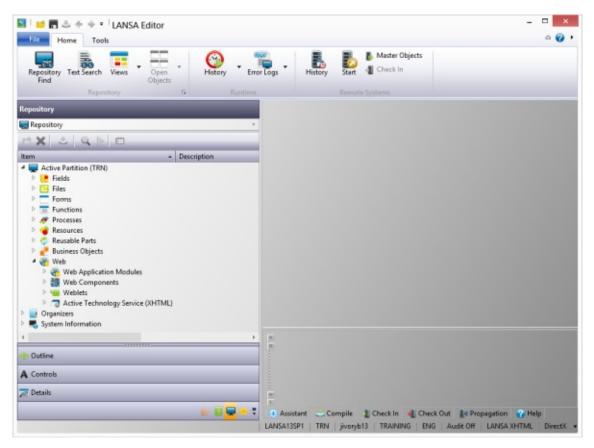
User ID: Password:	jivoryb13	
Use Window	ws credentials	
Partition	Current Language	Task ID
SYS	English	TRAINING
TRN	French	TS YS0001

Logon using the *User ID*, *Password*, *Partition* and *Task ID* provided by your course instructor.

If you are using a trial, stand alone installation, logon using:

Userid: **PCXUSER**, Password: **PCXUSER**, Partition: **DEM** and Task ID: **PCXTASK**.

The LANSA Editor will open. The appearance of the editor will depend upon the type of installation as well as the editor settings

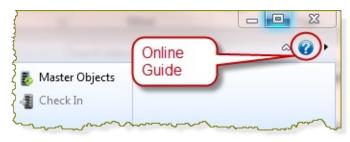


Note: Visual LANSA is shipped configured with a grey color scheme. You will find most of the images included in this tutorial use a blue color scheme. You will learn about changing the appearance of the IDE and editors in this tutorial.

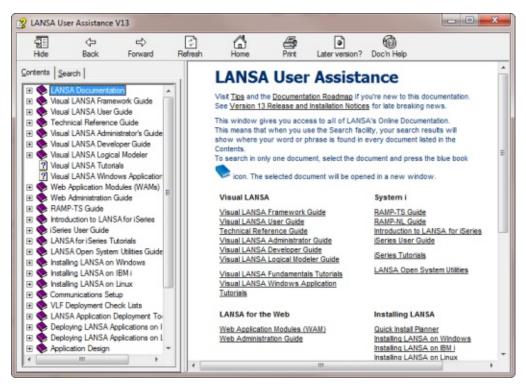
Step 2. Online Documentation

In this step, you will learn about the LANSA online help and documentation search facilities.

1.In the *LANSA Editor* click on the *Help* icon at the right hand side of the ribbon area..



The *LANSA User Assistance* contents page will be shown. You can use the *Search* tab from here to search for a topic in all LANSA guides.



Follow the numbered steps shown below on the left hand side. This is just one way to navigate through the online guide. Once you are experienced, you will find there are many ways to rapidly find the information you need.

1. Select the

Documentation

Roadmap link at the top of the right hand side.

2. Note that this takes you to links to sections of the guide for different types of user.

Select the *Developer* link

3. This provides more links to sections of the guide suitable for a developer.

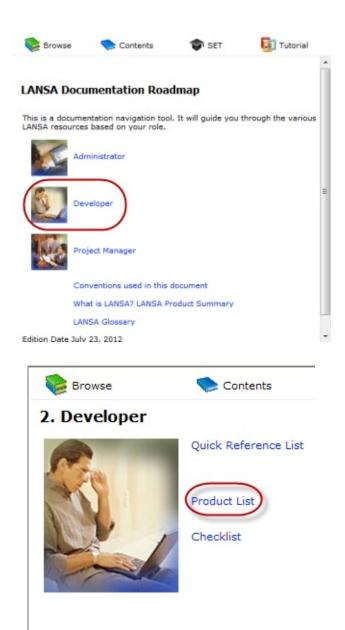
Select Product List.

LANSA User Assistance

Visit Tips and the Documentation Roadman you're new to this documentation.

This window gives you access to all of LANSA's Online Documentation. This means that when you use the Search facility, your search results will show found in every document listed in the Contents.

To search in only one document, select the document and press the blue book



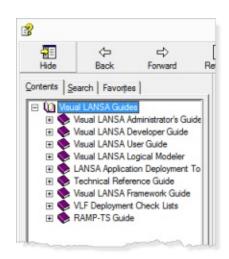
4. From here select *Visual LANSA Global* Guide link.

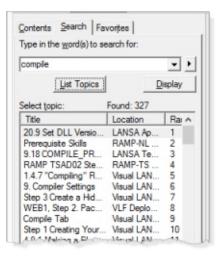


The Contents tab shows the guides grouped under the Visual LANSA Global Guide. This opens the Visual LANSA Guides as a separate document.

5. Switch to the *Search* tab, type in the word **compile** and select the *List Topics* button.

Note that there are around 300 topics containing the word compile. This is because you have searched all





the guides grouped under Visual LANSA Global Guide.

As you can see, this list is long and unwieldy. To make it more manageable, you can:

* sort the listed tems by topic title y clicking the *Title* olumn heading

* sort the items by *Location*, to split the ist by guide.

* narrow your earch using the earch features lescribed in bearching in the *Tips Guide*.

6. Select one of

the topics and select the *Display* button. Then select the *Contents* toolbar button

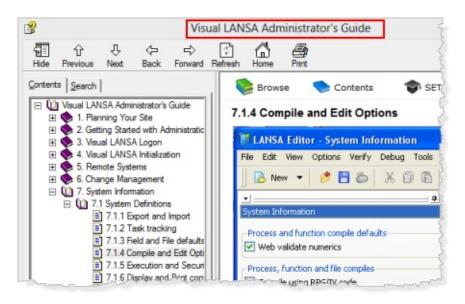
at the top of the displayed page.

Contents Search			
Type in the word(s) to se	earch for:		
compile		•	۲
List Topics	Di	splay	
Select topic:	Found: 327		_
Title	Location	Rai	*
RAMP TSAD08 Ste	RAMP-TS	72	
RAMP TSAD09 Ste	RAMP-TS	76	
9. Compiler Settings	Visual LAN	7	
7.1.4 Compile and E	Visual LAN	12	
1.3.2 Local Client or	Visual LAN	24	
5.4.1 Detailed Mess	Visual LAN	36	

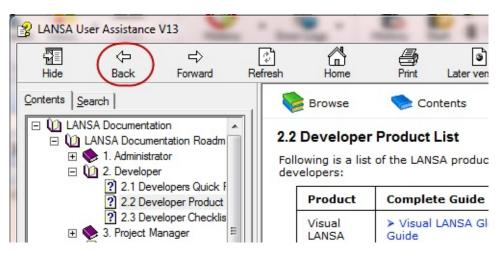


This will open the guide in which this topic was found.

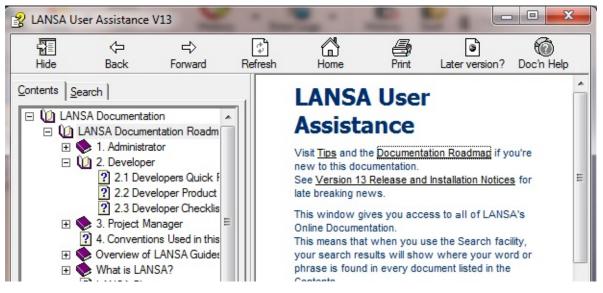
7. Switch to the Search tab and again list topics for the word **compile**. Note that there are around 19 topics found. This is because you are now searching for topics in one guide only. This is often the most efficient way to search for information.



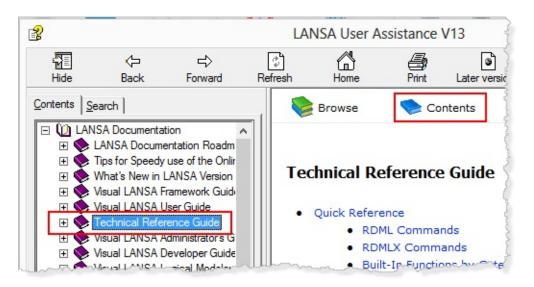
8.Close the individual guide you have just opened - *Visual LANSA Administrator Guide* in the example shown to return to *Visual LANSA Guides*. Close this guide, leaving the *LANSA User Assistant* guide open



9. Click the Back button (highlighted above) until you return to the *LANSA User Assistance* initial view.



10.Select the *Technical Reference Guide* and note the toolbar is shown for the selected guide – indicating that it may be opened as a separate guide using the *Contents* button. Most of the guides may be opened in this way.



Documentation Toolbar

11. The individual sections in the LANSA Guides display a common toolbar



To open a window displaying all the LANSA guides, press the *Browse* button in this toolbar. The *LANSA User Assistance* window will be displayed. You can use this window to search all LANSA guides



- 12. Note that you can also open the *LANSA Guides* from the ^{Documenta...} folder in the LANSA folder.
- 13. Of course the editor also has a menu button at the top right.

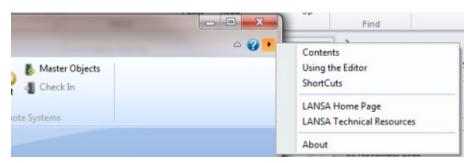
Contents open the full *Online Guide* reviewed above.

Using the Editor and ShortCuts are more specialized help files.

LANSA Home Page and *LANSA Technical Resources* are links to the LANSA web site.

About provides detailed information for your Visual LANSA software.

If you are making a support call you may be asked to use the *About* dialog to save information about your installed software and configuration.



14. The button highlighted below, hides or shows the ribbon area.



Step 3. F1 Context Sensitive Help

In this step, you will learn about the context sensitive help available throughout the LANSA interface.

If you are not logged on to the Visual LANSA Development Environment, log on now as described in *Step 2*. *Logon and Partition Initialization*.

1. Select the *Repository* tab. If you can't see the *Repository* tab, on the *Home* ribbon, select the *View* menu, then select *Repository*.

File Home Desig	jn Tools		
Repository Find	Views Open C Objects	Sompile Build History	Error Logs Execute
Repo	Views		
Favorites	Assistant (F4)	Details (F7)	Layout Helper
🖬 🗙 🕹 🔍 🕨	Breakpoints	Favorites (Shift+F8)	Outline (F6)
📳 Last Opened 🔚 Wel	Call Stack	Features (F2)	Propagation
DDRESS1		(P) readines (r2)	an ropagation
DEPTAB	E Check In	Go To (Ctrl+G)	Repository (F8)
iiDeptMnt			-
iiiCOM18	Check Out	Pelp	Text Search
🧔 IIICOM22	Sompile	Impact Analysis	## Variables
IIICOM23 IIICOM23 IIICOM23	A Controls	Import	Web Designs
iiiDeptCode	A	Sector Se	No besigns
iiiEmpEnguiry			10 million (1997)

Note that the *Repository* tab can also be displayed using the *F8* key.

2. With your cursor either on the tab or somewhere within the *Repository* tab, press F1.



The context help for the *Repository* tab is displayed in the *Help* tab. You can view all the text using the scroll bars. 3. Scroll to the bottom of the Repository Tab's text where you will see a topic with an up arrow n Standard Editor Tabs. If you click this topic, you will be taken to the documentation's next highest level for the current topic. From this higher level topic you can select any link to drill down into other related topics.

In this case, you can view all the Standard Editor tabs.

× #	4	Standard Edito	or Tabs			*
		Repository Tab	Details Tab	Debug Tabs	Propagation Tab	
		Favorites Tab	Source Tab	Compile Tab	Cross References Tab	
		Outline Tab	Assistant Tab	Check In Tab	Editor Features	
	6	Go To Tab Features Tab	Design Tab Repository Help Tab	Check Out Tab	Web Design Tab Editor Basics	
0 1	۲					-
1	Assist	ant 迭 Compile 🛛 🤇	🕜 Help			
			LANSA13 *S SYS	jivory13 *U	JII ENG Audit Courier	Colors LANSA XHTML

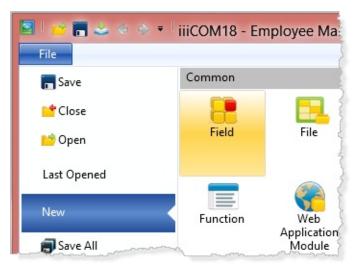
4. Click on one of the links. The page for that link will be displayed. Return to your original page by clicking on the 4 icon.

If these links provides insufficient information, you can open the guide containing the topic which you are viewing. In this example it is the *Visual LANSA User Guide*.

5. Click the blue book Scontents button to open the *Visual LANSA User Guide*. It will open at the help text topic you are looking at. In this example it will be the *Standard Editor Tabs*.

Y Visual LANSA User Guide 配 介 - 小 中 中				
Hide Previous Next Back Forward	Refresh Home Print			
Contents Search	Browse	Contents	SET SET	🛐 Tutorial
Visual LANSA User Guide	Standard Edito	or Tabs		
Editing Objects Standard Editor Tabs	Repository Tab	Details Tab	Debug Tabs	Propagation Tab
Editor Features	Favorites Tab	Source Tab	Compile Tab	Cross References Tab
	Outline Tab	Assistant Tab	Check In Tab	Editor Features
	Go To Tab	Design Tab	Check Out Tab	Web Design Tab
Details Tab	Features Tab	Repository Help Tab		Editor Basics

- 5. Close the *Visual LANSA User Guide* using the Windows *Close* votes.
- 6. From the *File* menu, select the *New* option, and select *Field*. Note that in this exercise you will NOT complete the create a new field process.



7. The *New Field* dialog will be displayed.

😬 New Field			×
Name		¥	Create
Description		▼	Cancel
Field Type	Alpha	•	
Field Length	10 ‡		Open in editor
Decimals	÷		Close
Reference Field			
Identifier			
Enabled For RDMLX			

8. Notice that the cursor is positioned on the *Name*. Press the F1 key to display the online help for *Field Name*.

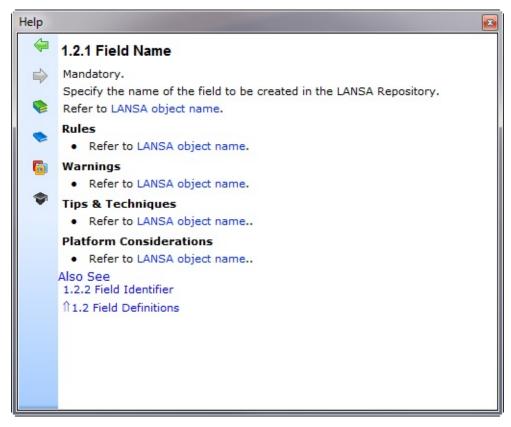
Help for *Field name* will be displayed in the *Help* tab.

	N.	ew Field				X		
	Nam	ie			•	Create		
	Desc	ription				Cancel		
	Field	Туре	Alpha		-	Cancel		
	Field	Length	10 ‡			Open in editor		
	Deci	mals	\$		1	Close		
	Refe	rence Field	-					
	Ident	tifier						
	Enab	oled For RDMLX	[]					
×	4	1.2.1 Field Na	ame					
		Mandatory.						
	٠	Specify the name Refer to LANSA		e created in the LANSA	Repository.			
	Rules Refer to LANSA object name.							
			NSA object name.					
	•	Refer to Warnings						
	•	Refer to U Warnings Refer to U	ANSA object name.					
	٠	Refer to Warnings	ANSA object name. ques					

Note: If the help text is larger than the *Help* tab's current size, you will not be able to enlarge the tab or use the scroll bars while the *New Field* dialog is open.

The *New* dialog is modal. You must either close the *New* field dialog and then enlarge the *Help* tabs area or enlarge the *Help* tab before using the *New* dialog..

You can also float any of the editor tabs as a separate window.



8. Close the *New field* dialog.

Summary

Important Observations

- When you logon to LANSA, you select the partition, language and task ID to be used. The LANSA Administrator will create profiles and authorities to control access to LANSA.
- LANSA can be started from the LANSA desktop folder or by using the Window's Start menu.
- The LANSA documentation contains a standard toolbar to access all the documents, or as single guides, SET and the Tutorials guides.
- The *Search* tab in a document is used to search the currently open guide or group of guides.
- You can find the location of a searched item in the *Contents* tab.
- The Browse 📚 button on the documentation toolbar opens the *User Assistance* window that allows you to search all LANSA guides.

Tips & Techniques

- Remember to use the F1 online help while using LANSA and these tutorials. There is online help throughout the user interface.
- LANSA provides a wide range of user, developer, administrator and technical documentation, as well as documentation by product.
- Use the *Tips Guide* to learn about the specific search techniques available with Visual LANSA.
- Use the *Contents* > toolbar button to open the specific guide whose topic you are reading. You can quickly review other topics in the guide or you can search within just one guide.
- Always open the contents list in the *Contents* tab before you do a search, if you want to find the location, in the documentation, of an item that has been found in a Search.
- If you want to narrow your help topics search to a single guide, simply use the *Contents* toolbar button to open that guide in its own window. Once the guide has opened, use the *Search* tab to search within the guide that is opened.

What I Should Know

• How to start Visual LANSA.

- How to logon using a specific partition, language and task ID.
- How to open the LANSA documentation and use the documentation toolbar.
- How to use F1 context sensitive help. (F2 context help will be tried later in this fundamentals tutorial.)
- How to search all LANSA guides or how to search a single guide.
- That you need to open the contents list before doing a Search, in case you want to find the topic in the content list.
- The initial set of personnel system demonstration objects in the Active Partition were loaded using the partition initialization.

VUI002 – LANSA Editor Parts

Objectives:

- To introduce the basic parts of the Visual LANSA Editor.
- To learn how flexible are the basic editor parts.
- To learn how to use autohide for the tab folders of the editor.
- You will not open any objects in LANSA. The focus of the exercise is how the parts of the interface can be changed. The purpose and features of each part are described in later exercises.

To achieve these objectives you will complete the following:

- Step 1. Editor Parts Overview
- Step 2. Repository, Favorite, Outline, Details Tabs
- Step 3. Docking/Undocking a Tab Sheet
- Step 4. Reset Editor Settings
- Summary

Before You Begin:

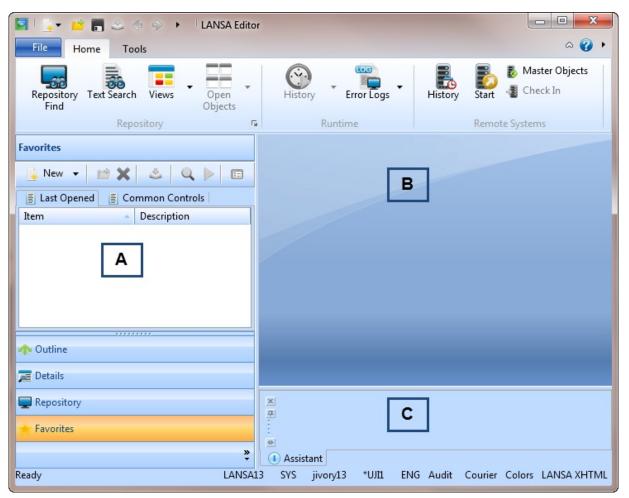
You may wish to review the following topics:

- Using the Editor in the User Guide
- Host Monitor in the Administrator Guide
- It is recommended that you complete the preceding tutorials before you start this tutorial.

Step 1. Editor Parts Overview

1. If you haven't already done so, start *Visual LANSA* as described in *Step 2*. *Starting the LANSA Development Environment*.

The initial appearance of the Visual LANSA Editor will depend upon your type of system installation. If you are using a *Slave Workstation* Visual LANSA System, the editor will initially appear something like the following:



- There are three main areas in the Editor:
 - A. The *Repository, Favorite, Outline,* and *Details* tabs. The *Repository* and *Favorite* tabs are used for selecting objects in LANSA. The *Details* tab is used to edit object details and the *Outline* tab lists the names of the objects open in the editor and shows the structure of these objects. These four tabs are initially situated on the left side of the editor but can be moved as required.
 - B. This area displays the details of the selected object. The tabs shown will

depend on the object selected. For example, if a Function is displayed, the tabs most likely displayed will be the *Source*, *Design*, *Repository Help* and Cross References tabs. If a file is displayed, then you will see these tabs: Fields in File, Logical Views, Rules and Triggers, Access Routes, Batch Control, File Attributes, Relationships, Cross References, Virtual Derivation.

C. The *Assistant, Help Text, Compile* (and optionally the *Check in, Check Out* and *Propagation*) tabs provide status information and assist in performing specific actions. They initially appear at the bottom right side of the editor.

The sample screens in this tutorial are from a *Slave Workstation* Visual LANSA system. Your screens will be slightly different if you are using an *Independent* Visual LANSA System.

Your editor's setup and system settings will impact the information that is displayed. Your screens may not be an exact match to the examples shown in these exercises.

2. You can resize these three main areas by dragging the horizontal or vertical splitters or dividers between the panes. Simply click on the divider (arrows will appear) and with the left mouse button pressed down, drag the divider to change the size of the pane.

IANSA Ed File Home Tools	ditor			→ □ → ×
Repository Text Search Views Open Find	• Example 1	Error Logs	History Start	Master Objects
Repository	G .	Runtime	Remo	te Systems
Favorites				
📱 Last Opened 📱 Common Controls				
Item Description		editor's par	or enlarge the nes using these viders	
normality of the second				
🔎 Details				
🖵 Repository	X 7		•	
	Assistant			
Ready LAN	ISA13 SYS jiv	ory13 *UJI1 EN	IG Audit Courier	Colors LANSA XHTML

Step 2. Repository, Favorite, Outline, Details Tabs

In this step, you will learn how to reposition the four basic editor tabs used with LANSA objects: *Outline, Details, Repository* and *Favorites*.

1. On the *Home* ribbon expand the *Views* menu to display all possible tabs. Note that a number of these tabs can also be opened using a function key.

File Home Desi	gn Tools		
Repository Text Search Find	Views Open Co Objects Co	Sompile ■ Build History	Error Logs Execute
Repo	Views		1
Favorites	Assistant (F4)	Details (F7)	Layout Helper
😐 🗙 l 😃 🖡	Breakpoints	Favorites (Shift+F8)	Outline (F6)
📳 Last Opened 🛛 🧺 Wel	Call Study	Easture (E2)	R & Descention
Item	Call Stack	Peatures (F2)	Propagation
 ADDRESS1 DEPTAB 	Check In	Go To (Ctrl+G)	Repository (F8)
iiDeptMnt iiiCOM18	Check Out	P Help	Text Search
IIICOM21 IIICOM22	Lompile	Impact Analysis	## Variables
 IIICOM23 IIIDepartments IIIDeptCode 	A Controls	Import	Web Designs
iiiEmpEnguiry			

By default, the tabs are shown using the *Tab View Style*, **Navigator**.

	~
💠 Outline	}
📜 Details	1
💂 Repository	× 単.
Favorites	
	1 - 5
»	
	1
Death	1
Ready	3

The Navigator style, shows tabs as buttons. These buttons may be hidden by using the Show Fewer Buttons option:

Favorites		i i i i i i i i i i i i i i i i i i i	
	» •	Show More Buttons	Ì
Ready		Show Fewer Buttons	
	_	Add or Remove Favorite	

For example:



You would then click on an icon to open a tab.

2. Right click on the tab title, and use the *Tab View Style* option to select *Standard*:

Favorites New Image: Constraint of the second sec	Max Layout Autohide Close	
Item	Tab View Style	Standard
		Navigator

The *Standard* tabs will be displayed at the bottom:



3. Right-click on any tab to open the context menu. Select *Top* to move the tab names to the top.

All four tabs will switch to the requested location. In the example below, the *Standard* tabs are now located at the top of the window:

Outline 📜 Details	💂 Repository 🔶 Favorites
🔓 New 👻 🖻 🗙	⊴ Q ▶ ⊡
📕 Last Opened 📑 Co	mmon Controls
Item 🔺	Description Mo

- 4. Click on the *Repository* tab, to display it.
- 5. Click on the autohide 😐 button in the corner of the tab to hide the *Repository, Details, Favorites* and *Outline* tabs.

When you use autohide, you have more room to work with the objects open in the editor.

- 6. Press F8 to see the *Repository* tab again. This will display the tab briefly and it will hide again when your cursor moves off it.
- 7. Click on another area of the editor and the tab folder is automatically hidden.
- 8. Click on any tab and then click on the Attach button **I** to lock the tabs back into place.

All tab folders are attached back to their original position and will now stay visible regardless of where the focus is in the editor.

Assistant, Help, Compile Tabs

These tabs appear at the bottom of the editor and you can maneuver them in the same way as for the Repository tab, except that you cannot dock them into another location.

Host Monitor Tabs

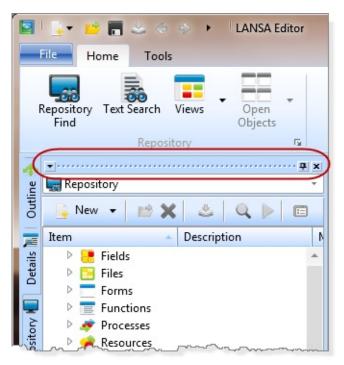
If you are using *Slave Workstation* Visual LANSA with a *Master* LANSA for iSeries System, the editor will include *Check in, Check Out* and *Propagation* tabs. You can undock them, adjust their size and hide them, as described for the *Repository* tab.

Step 3. Docking/Undocking a Tab Sheet

In this step, you will undock the *Repository* tab and move it to another position. Undocking is useful because it allows you to resize a dialog box to display more information without having to resize the panes of the window, which you have to do when a tab is docked.

Note that all tabs can be docked and undocked as described here.

1. With the *Repository* tab displayed, left mouse click on the docking bar and drag it to the center of the Editor's window.



2. Release the left mouse key when a title bar is added to the tab, or when it is the position where you want it to be.

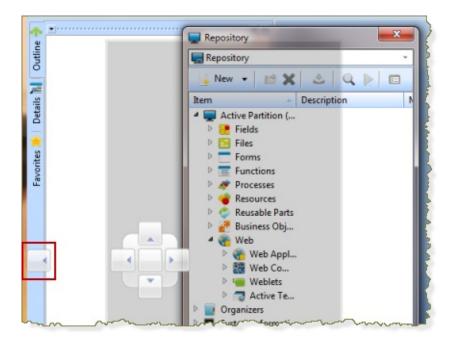
	ile Home Tools
R	epository Text Search Views Open Find Objects Objects
	Repository 🕞 Runtime
Outline 🧇	New P Repository Repository Repository
Details 📶	Item Descrip New - BX & Q F E
Detai	Item Description
avorites 🌟	 Files Forms Functions

The *Repository* tab is now undocked. It is now an independent window and you can position it anywhere on the screen, or on a second screen..

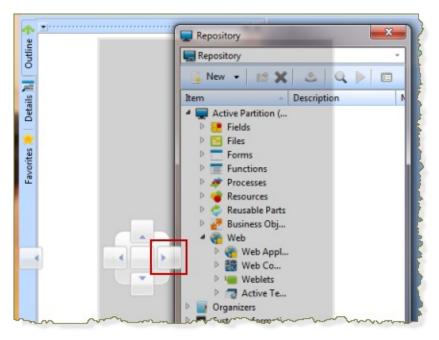
3. Resize the *Repository* tab so see all the columns.

Repository	-		-			×
🔙 Repository						4
📙 New 👻 📑 🗙	: 🕹 🔍 🕨 🗖					
Item 🔺	Description	Modified	Build Status	Details	Task	
Fields						
Files						
Forms						
Functions	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		

- 4. Close the *Repository* tab to hide it. By pressing F8, the *Repository* tab will reappear in the same location with the same size.
- 5. Double click on the title bar of the *Repository* tab and it will be returned to where it came from.
- 6. To re-attach the tab on the left, drag the *Repository* tab left, position your cursor on the *Left* button, and release the mouse button:



7. To have two tabs open side by side on the left, release the cursor over the Right button:

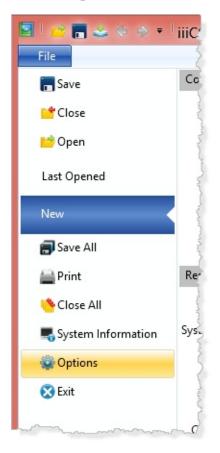


Your editor should then look like the following:

File Home Tools	\$	LANSA Editor	
Repository Text Search V Find Reposito	iews	Open Objects	History -
New New New New New New New New	Repository	Repository New Image: Constraint of the second	× Des

Step 4. Reset Editor Settings

1. On the *File* menu and click on *Options*:



2. Click the *Reset Editor* button to reset the Editor to its default settings. Later tutorials will illustrate how these settings can be set to your own preferences.



Summary

Important Observations

- If you need more room to work with objects open in the editor, you can use *Autohide* to hide the tab folders when they are not required.
- You can move undock individual tabs by dragging them from the docking bar.
- You can dock folders to either the left or right side of the editor.
- Folders can also be docked side by side.

Tips & Techniques

- Undocking tabs allows you to resize a window to display more information and it allows you to reposition the windows.
- Learn to use the function keys to quick redisplay tabs once they are closed.
- You may also dock/undock a tab by double-clicking on the docking bar.
- Using *Autohide* leaves you with more room to work with the objects open in the editor. This feature is extremely valuable when you are editing source code and want the maximum screen area for editing.

What I Should Know

- How to control the presentation of the basic editor parts.
- How to size the areas in the editor.
- How to turn *Autohide* on/off.
- How to dock/undock tabs.

VUI003 – Repository Tab

Objectives:

- To learn about the *Repository* tab.
- To learn about the types of objects listed in the *Repository* tab.
- To learn how you can customize the way the contents of the *Repository* tab are displayed.
- To learn how to open field definitions.

To achieve these objectives, you will complete the following:

Step 1. Repository Tab ContentsStep 2. Turn Alphabetical Grouping Off/OnStep 3. Arranging Columns in RepositoryStep 4. Object PropertiesStep 5. Accessing a Field

Summary

Before You Begin:

You may wish to review the following topics:

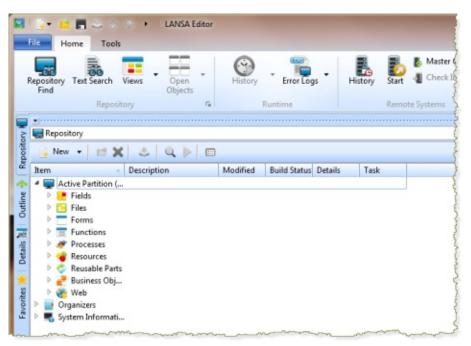
- Using the Repository Tab in the User Guide
- Alphabetical Grouping in the User Guide
- Information about Objects in the User Guide
- It is recommended that you complete the preceding tutorials before you start this tutorial.

Step 1. Repository Tab Contents

The *Repository* tab is the focal point for all development. It shows all objects in the active partition and provides access to objects in the LANSA system. You use the *Repository* Tab to open and work with objects, such as your application fields, files, forms and reusable parts.

1. Resize the *Repository* tab to see more of the columns of information available. **Reminder:** The columns that appear in the Repository tab will depend upon the type of Visual LANSA System that you have installed. The sample screens in this tutorial are from a *Slave Workstation* LANSA installation.

The *Repository* tab might appear something like the following:



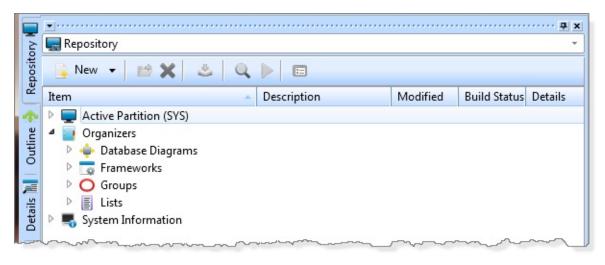
In this lesson, when you are asked to open an object, you will use the *Repository* tab to locate them.

2. Notice that there are three main groups of objects listed:

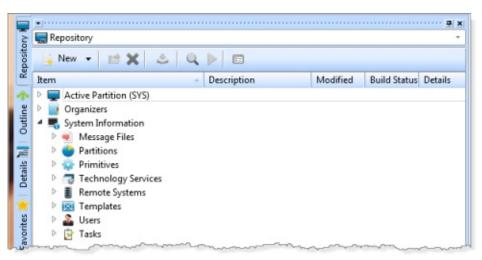
<i>Ac</i> tive Partition	The Active Partition stores all objects that a developer creates when building applications with LANSA.
Organizers	Organizers are objects that are used to group objects in the LANSA Repository.
System	System Information stores details about the Visual LANSA

Information installation.

3. Expand the *Organizers*. You will see a list of objects that you can use to group objects in the repository. For example, user defined lists and database diagrams are stored in *Organizers* along with *Frameworks* and *Groups*.



4. Expand *System Information*. LANSA Administrators will use this information to configure the development environment. Many of the settings here (for example Partitions) reflect setting established on the System i LANSA system.



5. Expand the *Active Partition(DEM)*. You will see a list of object type groups that you will use to create and build LANSA applications. A set of objects were imported into the repository when the partition was initialized.

5	Repository			
Repository	🔒 New 🝷 📂 🗶 🖉 🍳			
č	Item	Description	Modified	Build Status Details
3	Active Partition (SYS)			
ĕ	Fields			
Outline	Files			
0	Forms			
	1 official			
-	Functions			
ils m	roma			
Details 1	Functions			
Details In	 Functions Processes 			
orites 💛 Details 🐘	 Functions Frocesses Frocesses Resources 			

Step 2. Turn Alphabetical Grouping Off/On

In this step, you will learn how to group repository objects alphabetically.

1. On the *Repository* tab, expand the *Fields* node. By default the fields are grouped alphabetically.

Expand the letter A to see all fields starting with the letter A.

The list may appear something like the following:

🖬 ! 📴 📫 🐻 🍝 🔶 🕨 !	LANSA Editor				
File Home Tools					
Repository Text Search Views	Open Objects	Error	ogs H	listory Sta	art
Repository	5	Runtime		Rei	mote
Repository					
Repository					Ψ.
🔒 New 🔹 📄 🗶 💩 🔍					
Item	Description	Modified	Build Status	Details	Ta
4 🔚 Fields					*
A ABSOPT ADDRESS1 ADDRESS2	Action Bar Option Street No and Name	28/11/19		Alphanu Alphanu	
ADDRESS2 ADDRESS3 Barrow	Suburb or Town State and Country	28/11/19 28/11/19		Alphanu Alphanu	~~~

The fields shown are used in the Personnel System demonstration.

2. Right-click on the *Fields* node and deselect the *Alphabetical Grouping* option from the context menu.

Repository		(
🔙 Repository		
🔒 New 👻	🖻 🗙 🕹 🔍 🕨 🗉	(
Item	 Description 	on ⁴
4 🤐 Fiel -	Find	
	Is Favorite	Option
	Alphabetical Grouping	nd Name

The grouping will be turned off.

3. All fields are now displayed in alphabetical order but without alphabetical grouping.

ne Re	epository					Ŧ
14 I	New - 时 🗙 🙁 🔍					
ltem		Description	Modified	Build Status	Details	Т
	📒 Fields					
	@@RRNO @@RRNO	Relative record number			Packed(1	
	e e @@UPID	Field update / access i			Packed(7	
	ABSOPT	Action Bar Option			Alphanu	
	ADDRESS1	Street No and Name	28/11/19		Alphanu	
	ADDRESS2	Suburb or Town	28/11/19		Alphanu	
	ADDRESS3	State and Country	28/11/19		Alphanu	

Alphabetic groupings can be used with most objects listed in the Active Partition.

Step 3. Arranging Columns in Repository

In this step, you will change the order of the columns on the *Repository* tab, resize and hide a column.

1. Click on the *Details* column heading and drag-and-drop it the left of the *Modified* column.

Repository						
🔙 Repository						*
일 New 👻 📑	K & Q ▶	•				
Item	DetBitscr	ipti on	Modified	Build Status	Details	Ta
4 📜 Fields		-		-		*
RRN @	D Relati	ve record number	1		Packed(1	
🖻 📑 @@UPID	🛨 Field	update / access i.			Packed(7	
	Actio	n Bar Aptian	m	m	Alphapu	~
Repository	1					
					1.1.1	
Item		 Details 	Description	M	odified 💈	
🖉 4 🔚 Fie	lds				7	
Þ 📒	@@RRNO	Packed(1	Relative record	number	100	
▶ 📑	@@UPID	Packed(7	Field update / a	ccess i	Š	
⊳ 📒	AB\$OPT	Alphanu	Action Bar Opti	on	2	
		Alphanam	Action bai opti	011		
	ADDRESS1	Alphanu	Street No and N		/11/19\$	

2. Right-click the column header area and deselect the *Modified* column in the context menu.

Repository											
Repository											,
🔓 New 👻 时 🔀	2	Q									
ltem		-	Details	Descript	ion		Modified	Build	Status	Task	_
4 💓 Fields					~	Item					
Ø@RRNO			Packed(1	Relative	~	Descriptio	on				
🖻 😬 @@UPID			Packed(7	Field up	~	Modified					
ABSOPT			Alphanu	Action E	-	Build Stat	us				
ADDRESS1			Alphanu	Street N	-	Details					
ADDRESS2			Alphanu	Suburb	-	Column	lame				
ADDRESS3			Alphanu	State an		Туре					

The *Modified* column is now hidden.

- 3 Right-click the column header area and select the *Modified* column again to make it appear. Drag the *Modified* column to the right of the *Description* column.
- 4. You can sort the repository objects by clicking on the column heading. Click on the *Details* column heading to sort the list by the field type.

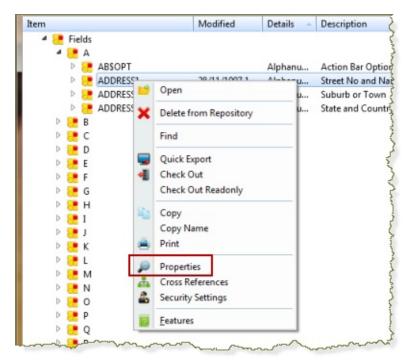
🔜 Repository			
🔋 New 👻 📂 🗙 🛛 🔕			1
Item	Modified	Details 🔺	Description
4 😬 Fields			
🖻 🔚 BLANK		Alphanu	Blank / blanks variable
BLANKS		Alphanu	Blank / blanks variable
DF_ELHLP1		Alphanu	Help for function DW
DF_ELOPER		Alphanu	Operation (+ - x /)
DE EL SELT		Alahan	Select

- 5 Click on the *Item* column heading to sort the fields alphabetically again.
- 6. Reset *Fields* to *Alphabetic Grouping* again.
- 7. Close the *Fields* node.

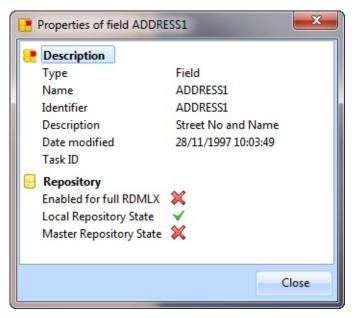
Step 4. Object Properties

In this step, you will view the properties of an existing field, ADDRESS1 using the *Repository* tab.

1. Open the *Fields* node and select the ADDRESS1 field from the A field group list. Right click on the ADDRESS1 field. Notice the list of options available in the context menu.



2. Select the *Properties* option. A brief summary of field properties will be displayed.



3. Close the properties dialog.

In the next step, you will open an object in the editor.

4. Before you open the object, select the *Outline* and *Details* tabs. (If these tabs are not open, select them from *Views* in the *View* menu or by pressing function keys F6 and F7.)

Notice that these tabs are blank because no objects are currently open.



If necessary, shrink the width of this tab so that you can see the Edit view in the right hand pane in the following steps.

Leave these tabs open for the next step, *Step 5*. *Accessing a Field*.

Step 5. Accessing a Field

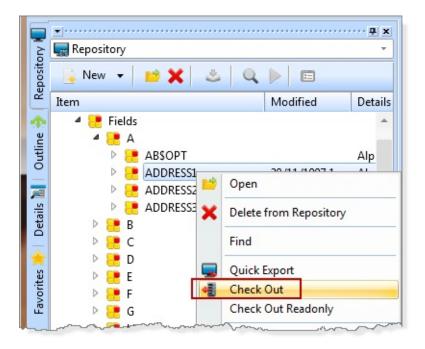
In this step, you will open and view an existing field, ADDRESS1 using the *Repository* tab.

- 1. In the *Field* group in the *Repository* tab, right-click the field ADDRESS1 and choose the *Open* option from the context menu. Alternatively double click on an object to open it in the editor.
- 2. When you attempt to open an object in LANSA, a number of checks will be performed. If you are not authorized to an object or if another developer has already opened an object, the following message will be displayed. As noted, you can then open the object in read-only mode.

Read Only	warning on ADDRESS1 (Field)
?	This object was not checked out from the central repository for update. You may open this object only in read-only mode. Continue?
	Yes No

3. If you are using an Independent Visual LANSA, with user id PCXTASK, you will be able to open the field for editing.

If you are using a Slave Workstation Visual LANSA, together with a normal developer user profile, you may need to check out the field definition from the iSeries master.



If you are authorized to the field, it will be opened for editing in the LANSA Editor.

4. Display the *Details* tab. The contents of this tab will change as you move from one part of the field to another.

Re	pository Text Search Find		Open Objects	Hi	story Error Log	F	Propagate Designer	Paste Cut Paste Copy Clipboard	Find	Remote Systems •	
			@ x	Definition	Rules and Trigger	Visualiza	tion Reposi	tory Help Cross Ref	erences		1
N Id Fi	ame	ADDRESS1									
is la	lentifier	ADDRESS1		4 🔚 De	tails						
F	eld Type	Alpha			me		ADDRESS1				
	ield Length	25	÷	Ide	entifier		ADDRESS1				
	-	D		Typ	De .		Alphanum	veric			
D	ecimals		0		ngth		25				
B D	efault Value	*BLANKS			cimals						
R	eference Field			De	fault Value		*BLANKS				
	escriptions			Ref	ference Field						
-				0.	and all and		Street No a				
D	escription	Street No and	Name	Lat	scription		Address 1.				
F	ield Label	Address 1			ading 1		Address I. Address lir				
0	olumn Headings	Address line 1			ading 2		Auditess III				
C	orannin readings	Piddress mie I			ading 3						
-											
					it Mask						
D	etails				yboard Shift						
E	dit Mask	1			abled For RDMLX		No				
		-	_		stem Field		No				
K	eyboard Shift		•		ompt Process						
E	nabled For RDMLX				ompt Function as Name						
S	ystem Field			All	as Marrie						
P	rompt Function	-		4 🥪 Ing	put Attributes						
			_	FE			Field exit k	ey required		Func	ti
	rompt Process			LC				entry allowed		Func	ti
A	lias Name			-							
				4						•	
			Γ	× ,0	Job Status		Results			1	De
				-	Completed		0 fatal errors	s - 0 warnings			AD
				×				0.000000000000000000000000000000000000			

You may wish to close the *Repository* tab, (you can use F8 to redisplay). Using the skill learned in the *VUI002 - Editor Parts* exercise, you should be able to resize the different area of the editor to layout your workspace to meet your personal preferences. You may wish to turn on *Autohide* for the bottom tabs.

- 5. Click on the *Input* attributes and then the *Output* attributes to see the changes in the *Details* tab.
- 6. Select the different tabs, *Definition*, *Rules and Triggers*, *Visualization*, *Repository Help* and *Cross Reference* to view the details of the field definition.

Do not make any changes to this field.

7. Close the field 🖄 in the editor.

Summary

Important Observations

- The *Repository* tab is the starting point for most of your work in the editor. You can access all objects in LANSA from the *Repository* tab.
- Once you have opened an object it will be added to the *Last Opened* list on your *Favorites* tab.
- You can turn on or off the alphabetical grouping of objects in the *Active Partition*.
- The columns in the *Repository* tab can be reordered and hidden.
- You can sort object lists using the column headings.
- You initially open objects for editing using the *Repository* tab.
- The initial set of personnel system demonstration objects in the Active Partition were loaded using the *Partition Initialization*, available from the *Visual LANSA Logon* form.

Tips & Techniques

• You can undock the *Repository* tab to maximize the size of the window. It can be closed and easily redisplayed using F8.

What I Should Know

- What information can be accessed from the *Repository* tab.
- How to dock and undock tabs.
- How to turn alphabetical grouping off.
- How to change the order of columns in the *Repository* tab.
- How to hide columns in the *Repository* tab.
- How to open fields using the *Repository* tab.

VUI004 - Details, Outline, Favorites Tab

Objectives:

- To understand the purpose of the *Details* tab.
- To understand the purpose of the *Outline* tab.
- To understand the purpose of the *Favorites* tab.
- To learn about feature help (F2) for objects.
- To learn to use the *Go To* tab to view the errors in the object currently open.
- To learn to view the error message of a field in error.

The focus of this exercise is how you use editor features to work with repository fields and files. The focus is not on the definitions of these objects. The *LANSA Repository* and *Form Development* Tutorials provide detailed training about these objects.

To achieve these objectives, you will complete the following:

- Step 1. Copy the ADDRESS1 Field
- Step 2. Details Tab
- Step 3. View Errors
- Step 4. Open Another Object in Editor
- Step 5. Outline Tab
- Step 6. Favorites Tab

Before You Begin:

You may wish to review the following topics:

- Using the Outline Tab in the User Guide
- Using the Details Tab in the User Guide
- It is recommended that you complete the preceding tutorials before you start this tutorial.

Step 1. Copy the ADDRESS1 Field

In this step, you will create a new field, *iiiAddressLine1* by copying the ADDRESS1 field so that you can open the new field for editing. You will also copy the rules, visualization and help text from the ADDRESS1 field.

1. Using the *Repository* tab, right click on the **ADDRESS1** field and select the *Copy* option from the context menu.

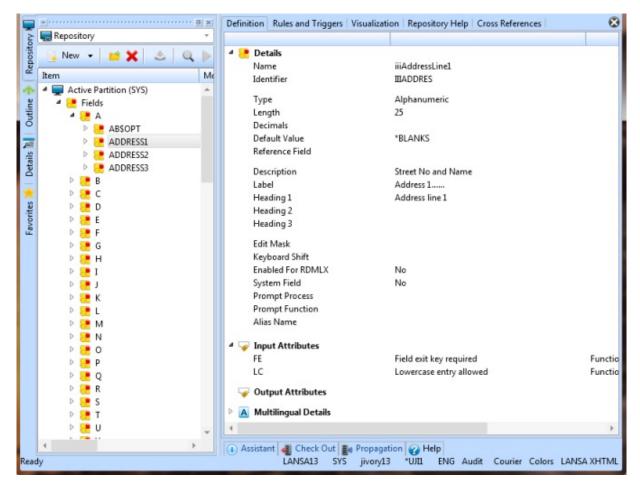


2. The Create as a copy of ADDRESS1 dialog will appear.

Create as copy of A	DDRESS1	×
Name	iiiAddressLine1	Create
Description	Street No and Name	Cancel
Initial Public Access	Normal 👻	
Identifier	IIIADDRES	Open in editor
Copy rules and trigger	rs 🔲	
Copy visualization		
Copy help text		

- a. Enter a *Name* of **iiiAddressLine1** (where **iii**=your initials).
- b. Note that the field has been given an *Identifier* of IIIADDRES
- b. Select the *Copy rules and triggers* option.
- c. Select the *Copy visualization* option.
- d. Select the *Copy help text* option.
- e. Select the *Open in editor* option.
- f. Press the *Create* button.

The iiiAddressLine1 field will be opened in the editor.



Step 2. Details Tab

In this step, you will review the repository field definition for the iiiAddressLine1 field to see how the different tabs are used to edit objects. The *Details* tab is used to display and edit selected properties of objects.

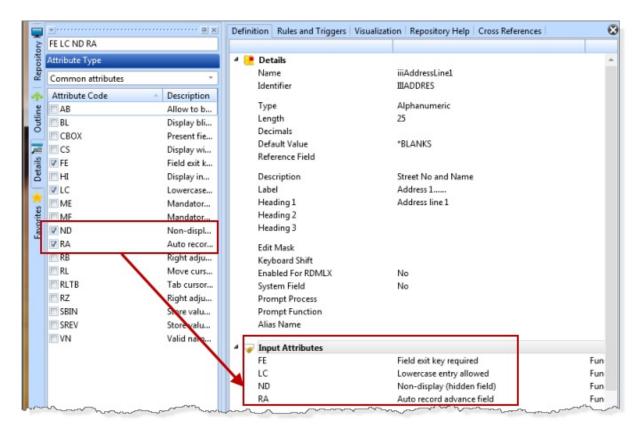
1. Select the *Definition* tab. You can edit most of the basic field characteristics using this tab.

To change the list of *Input* and *Output* attributes, you must use the *Details* tab.

-			bennition nares and miggers in	sualization Repository Help Cross References	8
S	FELC				
osito	Attribute Type		4 😬 Details		
Repository	Common attributes		Name	iiiAddressLine1	
-			Identifier	IIIADDRES	
+	Attribute Code	Description	Type	Alphanumeric	
Outline	AB	Allow to b	Length	25	
6	BL	Display bli	Decimals	25	
	CBOX	Present fie	Default Value	*BLANKS	
Details 📗	CS CS	Display wi	Reference Field	DEATING	
ails	V FE	Field exit k	Reference rield		
Det	III HI	Display in	Description	Street No and Name	
	V LC	Lowercase	Label	Address 1	
-	I ME	Mandator	Heading 1	Address line 1	
uje.	MF	Mandator	Heading 2		
Favorites	ND	Non-displ	Heading 3		
	RA	Auto recor	Edit Mask		
	RB	Right adju	Keyboard Shift		
	RL	Move curs	Enabled For RDMLX	No	
	RLTB	Tab cursor	System Field	No	
	RZ	Right adju	Prompt Process	140	
	SBIN	Store valu	Prompt Function		
	SREV	Store valu	Alias Name		
	VN	Valid nam	Allas Name		
	VIN	valid nam	4 🥪 Input Attributes		
			FE	Field exit key required	Functio
			LC	Lowercase entry allowed	Functio
			Output Attributes		
			Multilingual Details		
			•		÷.
			🚯 Assistant 🐗 Check Out 👔 P	Propagation 🕢 Help	
Read	у				s LANSA XHTML

Click on the *Input attributes* list. The *Details* tab will be displayed as shown:

2. Select some of the *Input* attributes and you will see that they are immediately included in the list of attributes in the *Definition* tab.



- 3. Deselect the extra *Input Attributes* that you selected.
- 4. Select the *Rules and Triggers* tab.

This tab simply displays the rule or triggers details. You cannot edit any of the information from this tab. Information must be selected and edited from the *Details* tab.

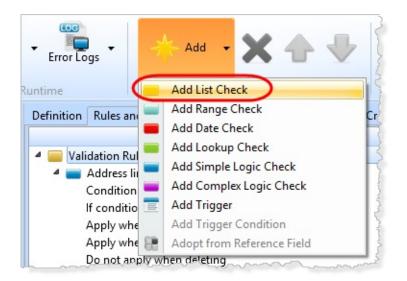
Expand the list entry Address line 1 cannot be blank. Notice that the *Details* tab is now shown on the left:

	<u>*</u>]······		Definition Rules and Triggers Visualization Repository Help Cross Referen	ces 🛛 😵
ory	Description	Address line 1 cannot be blar		
Repository	Sequence	1 ‡	4 🧱 Validation Rules	
Rep	Rule Definition		Address line 1 cannot be blank	
-	#iiiAddres *ne *blanks		Condition - #iiiAddres *ne *blanks If condition is found to be true NEXT else ERROR	
ne			Apply when inserting (ADD)	
Outline			Apply when updating (CHG)	
Ě			Do not apply when deleting	
1			Message - DC@M01 - DEM0044 - Address line 1	
Details 📗 Outl				
e				
*				
orite				
Favorites	Validation Usage			
	7			
	When Inserting	Always apply rule (ADD) -		
	When Updating	Always apply rule (CHG) -		
	When Deleting	Never Apply Rule *		
	Actions			
	Condition Is True	Evaluate next rule (NEXT) *		
	Condition Is False	Set field in error (ERROR) *		
	Error Message			
	Message Number	DEM0044		
	Message File	DC@M01		
-	Message Text		man and the second and the second sec	

5. Change the *Description* of the rule in the *Details* tab to say **No blanks allowed**. Notice that the rule Description in the Rules and Triggers tab is updated as you type.



6. On the *Home* ribbon, expand the *Add* menu, and select *Add List Check*.



Notice that the details for the new rule are entered using the *Details* tab. Do not enter any values.

A number of red triangles have appeared in the interface. If you click on these triangles, an error message will be displayed:

Definition	Rules and Triggers	Visualization	Repository Help	Cross References	8
	alidation Rules				
At lea	st one value must be	defined for a v	alue check rule.		
_	Condition - #iiiAddre	s *ne *blanks			
	If condition is found	to be true NEX	Felse ERROR		
6	Apply when inserting	(ADD)			
	Apply when updating	g (CHG)			
	Do not apply when d	eleting			
	Message - DC@M01	- DEM0044 - Ad	ddress line 1		
	New list check				
At	least one value must	be defined for	a value check ru	ile.	
how		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	-man		~

Leave this field definition open, and the errors will be discussed *Step 3*. *View Errors*.

Step 3. View Errors

In this step, you will use the *Go To* tab to quickly locate errors and view the messages associated with an error.

An object cannot be saved if it has errors.

In Step 2. Details Tab , you added an incomplete rule to the iiiAddressLine1 field.

- 1. With the **iiiAddressLine1** field still displayed in the editor, select the *Definition* tab.
- 2. Double click on the *Default value* in the *Definition* tab.
- 3. In the *Details* tab, change the *Default value* of the field to **xxx**. This will cause an error. Red triangles will appear.
- 4. Press the Save button on the editor toolbar.



An error message will be displayed.



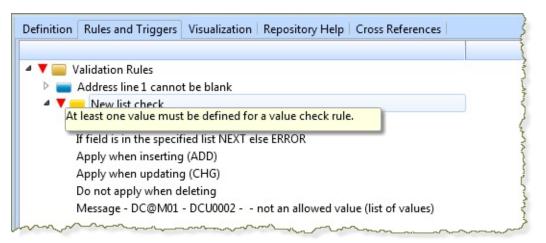
- 5. Click OK.
- 6. The *Go To* tab will automatically display showing all error messages related to the object. (The *Go To* tab can also be displayed by pressing Ctrl + G.)

File Home	🛎 🍖 🌩 🔸 Tools	IIIAddressLine	l - Stree	t No and Name - LANSA E	ditor"			
Repository Tex Find	Search Views	Open Objects		tory Error Logs	Designer	Clipboard	Find	Remote Systems
			- ₽ x	Definition Rules and Tr	iggers Vis	ualization	Reposit	ory Help
È 🖗 🗉 🕶								
Categories			-	4 🔻 📑 Details				
A K Error				Name			ddressLi	nel
-	ast one value must	he defined for as	alu.	Identifier		ША	DDRES	
	e xxx is empty, inval			Туре		Alp	hanume	eric
	e look is empty, inval	a of not anowed		Length		25		
u III Valu				Length		25		

7. Double-click on the red triangle of the error message text "*At least one value must be defined for a rule check*."

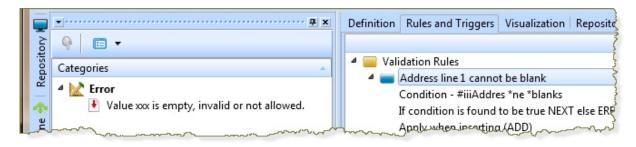
Notice that the *Rules and Triggers* tab is automatically displayed and the rule in error is selected.

8. Select the Rules and Triggers tab. Click on the E T In New list check line to display the error details.



- 9. Press the delete rule 💥 on the Home Ribbon to remove the incomplete validation rule.
- 10. Select the *Go To* tab (or press **Ctrl+G**).

Notice that, once the rule is deleted, the error is no longer listed.



- 12. Select the *Definition* tab. Click on the *Default Value*. The *Detail* tab will be displayed.
- 13. Enter a value of '**xxx**' (with single quotes because this is an alphanumeric field). The error message will disappear.
- 14. Close the field definition using the *Close* option from the *File* menu. When prompted to save the field changes, select *Cancel*.

ressLine1?	
No	Cancel

15. The field definition of iiiAddressLine1 should remain open for the next step, *Step 4. Open Another Object in Editor.*

Step 4. Open Another Object in Editor

In this step, you will open and view an existing File, the Employee Master file PSLMST, from the *Repository* tab. This means opening a second object (the first is the field **iiiAddressLine1** which you used in *Step 2* and *Step 3*. By opening this second object, you will learn how to switch between objects open in the editor.

- 1. On the *Repository* tab, expand the *Files* list.
 - a. Expand the P list. (If you have turned off Alphabetical Grouping, skip this step. Refer to *Step 2*. *Turn Alphabetical Grouping Off/On*.)
 - b. Right-click file **PSLMST** and choose the *Open* option from the context menu, to open the file for editing. (You can also open the file by double clicking on the file's name in the *Repository* tab.)

Note: You will not be changing this file in this tutorial, so it can be opened as read-only if it is checked out read-only, or if you are not authorized to use the file.

Home Tools	IIII Build	🛞 - 📸 - Togs - Togs - Remote	
Repository 12	Compile 5	Runtime Systems •	
■ Repository =		ules and Triggers Access Routes Batch Control File	
New ▼ ■ X & Q ▶ □	Field Name Primary Keys V > EMPNO	Description Employee Number	Ref. Field Type
Item Modified	4 V Real Fields	Employee Number	Alpha
→ These → → □ A → □ B → □ C → □ C → □ C → □ C → □ C → □ C → □ C → □ H → □ H → □ H → □ K → □ K → □ M → □ N	Constant of the second se	Employee Number Employee Sumame Employee Siven Name(s) Street No and Name Suburbo T Tawn State and Country Post / Zip Code Home Phone Number Business Phone Number Start date (YYMMDD) Termination Date (YYMMDD) Department Code Section Code Employee Salary	Alphi Alphi Alphi Alphi Alphi Signe Alphi Signe Signe Alphi Alphi Alphi Pack
▶ ■ ▶ ■ ▶ ■ ▶ ■ ▶ ■ ■ ■	PJFs Before Read Virtuals	Start Date (DDMMYY) Termination Date (DDMMYY) Monthly Salary	RETDAT Signe RETDAT Signe SALARY Pack

- Click on the different tabs to view the details of the file definition.
 Do not change any file details. Leave the PSLMST field open in the editor.
- 3. Notice that the *Previous* and *Next* buttons are enabled on the editor toolbar.

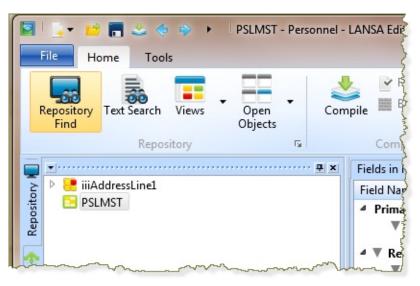


These button switch between the objects that are currently open in the editor.

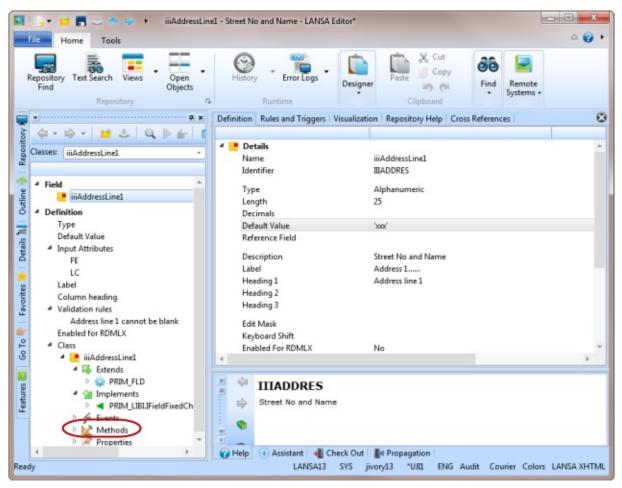
Step 5. Outline Tab

In this step you will learn about the *Outline* tab. The *Outline* tab is used to view and switch between objects that have been opened for editing. The *Outline* tab is also used to navigate within an object. It displays information about the structure of an object and will display feature help for object properties.

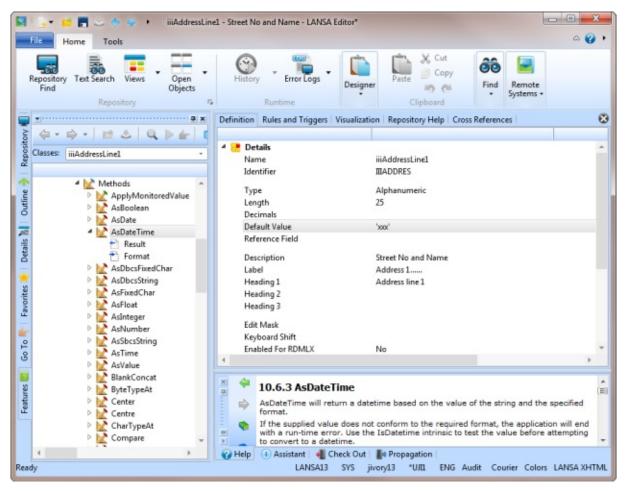
1. Select the *Outline* tab. The fields iiiAddress1 and the file PSLMST are shown:



- 2. Click on **iiiAddressLine1**. Notice that the field definition is now shown in the editor.
- 3. Click on **PSLMST**. Notice that the file definition is now shown in the editor.
- 4. Click on **iiiAddressLine1** again.
- 5. While iiiAddressLine1 is highlighted, press F2.Features of iiiAddressLine1 are displayed in the *Features* tab.Default help text is displayed in the *Help* tab.



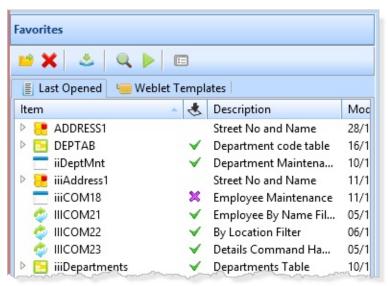
- 6. Expand the *Methods* of **iiiAddressLine1** in the *Features* tab.
- 7. One by one, double click on some of the Methods of the iiiAddressLine1 field. Help information relating to each Method (context help) will be displayed in *Help* tab at the bottom of the editor.



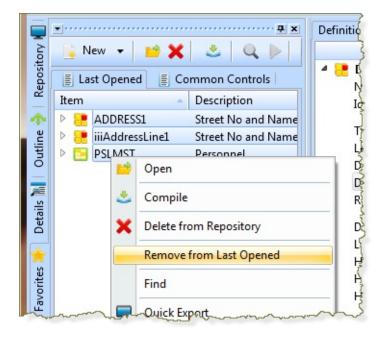
You may resize the Help pane, or float it as a separate window if you wish.

Step 6. Favorites Tab

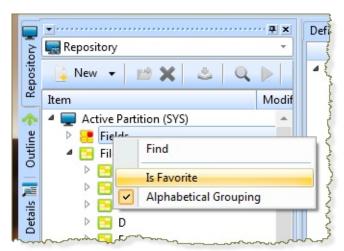
In this step, you will view the *Favorites* tab By default the *Favorites* tab contains the *Last Opened* tab and the *Weblet Templates* tab if your partition is web enabled. The *Favorites* tab can contain any other tab which you add to *Favorites*.



1. To remove objects from the *Last Opened* list, select them and use the right mouse menu:



2. In this step you will add a list to your *Favorites* tab. Switch to the *Repository*

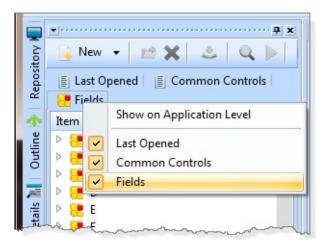


tab. Select the *Fields* list and use the right mouse menu option *Is Favorite*:

3. Switch to the *Favorites* tab. The *Fields* list is now shown on the *Favorites* tab.

Favorit	tes			
12 >	< ≗ Q			
≣ La	st Opened 🍋 V	Veblet Templates 🛛 📒 Fie	lds	
ltem		Description	Modified	Bu
4 💽	A			
⊳	😬 AB\$OPT	Action Bar Option		
⊳	📒 ADDRESS1	Street No and Name	28/11/19	
⊳	📒 ADDRESS2	Suburb or Town	28/11/19	
⊳	📒 ADDRESS3	State and Country	28/11/19	
⊳	😬 AudDate	Audit Date	15/10/20	
⊳	📑 AudDateX	Audit Date	15/10/20	

4. Use the right mouse menu to remove the *Fields* list from your *Favorites* tab:



Often you will create of *Editor Lists* of objects you want to work with and make these lists favorites.

Summary

Important Observations

- The properties of objects are edited in the *Details* tab.
- You move between open objects using the *Outline* tab.
- *Feature* help is used to display detailed information about objects.
- Errors are marked with a red triangle. (Warnings are marked with a brown triangle)
- You can view the message for an error or warning by clicking on the triangle marking the error.
- Use the *Go To* tab to view all errors in the open object. To locate the context of the error, double-click on the error message.
- The *Favorites* tab may contain many different lists of objects. It provides a fast way to access commonly used objects.
- The *Is Favorite* context menu option is used to add a list to the Favorites Tab. Whenever you see this option in a context menu, then the related list can be added to the *Favorites* tab.

Tips & Techniques

- The keyboard shortcut for displaying the *Go To* tab is **Ctrl** + **G**.
- The keyboard shortcut for displaying the *Feature* help is **F2**.

What I Should Know

- How to use the *Details* tab to edit objects.
- How to use the *Favorites* tab.
- How to use the *Outline* tab to switch between objects.
- How to display feature help for objects.
- How to use the *Go To* tab to view the errors in the object currently open.
- How to view the message for an error or warning.

LANSA Editor Tutorials

What are the LANSA Editor Tutorials?

This tutorial is for new LANSA developers and is designed to teach basic source code editing skills. It focuses on the editor features and techniques for using the Visual LANSA source code editor as opposed to teaching programming skills to build applications. No experience with the LANSA Repository or RDML programming language is required.

This brief set of exercises covers the essential areas of the LANSA editor:

VED010 - Format Source Code

VED020 - Edit Source Code

VED030 - Auto Complete and Command Assistant

VED040 - Execute Applications

Before you Begin

You must have LANSA Demonstration Personnel System installed in the partition that you will use with the set up options as described in What Partition Should I Use?

The LANSA Demonstration Personnel System contains all the objects used by these exercises.

Tips for using the exercises

- Check off each step in the exercise as you complete it.
- Follow the instructions very carefully.
- Remember to replace iii with your unique 3 characters. You will not always be reminded to make this substitution.
- These exercises assume that you have not previously customized the editor interface. If you have already customized your environment, the example screens and instructions may not exactly match your customized development environment.

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

Later exercises are designed to use skills from the earlier exercises. These exercises are designed to be completed in sequence.

VED010 - Format Source Code

Objectives:

- To learn how to copy a form and open it in the editor.
- To learn how to create a process and function and open them in the editor.
- To learn how to show and hide line numbers and indentation of the source code.
- To learn how to change the way statements are formatted.
- To learn how to hide DEFINE_COM statements in component source code.
- To learn how to use *F2 Features* help.
- To learn how to compile an object and display the compile message details.

The focus of this exercise is to control the appearance of code in the editor. The purpose of the programs and the actual meaning of the RDML commands is not important. The RDML Programming tutorials teach the basic coding practices.

To achieve these objectives you will complete the following:

- Step 1. Create a Copy of Form XDXSettingsDialog
- Step 2. Create a Process and a Function
- Step 3. Turn on Autohide
- Step 4. Change Formatting Options
- Step 5. Editor Source Settings
- Step 6. Word Wrap
- Step 7. Submit a Compile
- Step 8. Display Error Log
- Step 9. Display Feature Help Text
- 4.Expand Methods and double click on any of the component's methods. Help text is displayed explaining how the selected method can be used.

Before You Begin:

You may wish to review the following topics:

- *Repository Tab* in the Visual LANSA User Guide
- *Favorites Tab* in the Visual LANSA User Guide

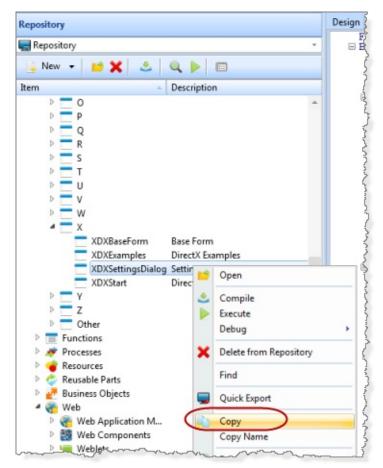
In order to complete this tutorial, you must have completed all the previous exercises in this workshop.

Step 1. Create a Copy of Form XDXSettingsDialog

In this step you will create a copy of the XDXSettingsDialog form for editing. Forms are used to create graphical Windows applications. Forms have a design layout (a sample of the form design), as well as RDMLX code that is used to control the behavior of the form. You will use this sample form to learn the basics of controlling the display of LANSA code in the editor. The actual meaning of the code is not important at this stage.

Creating a copy of form XDXSettingsDialog will enable you to edit the form.

- 1. On the *Repository* tab, expand the *Forms* node.
- 2. Expand the **X** node to see a list of all forms starting with the letter **X**.
- 3. Right-click form **XDXSettingsDialog** and choose the *Copy* option from the context menu.

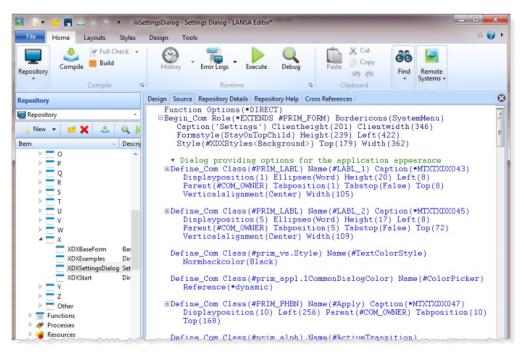


4. The *Create as copy of XDXSettingDialog* dialog will appear. Enter a *Name* of **iiiSettingsDialog** (where iii=your initials) or DEMCOM01 if you are using a trial version of Visual LANSA.

Note that a *Framework* of **Personnel & Payroll** has also been selected. Components are organized into *Frameworks*. You can change a component's *Framework* setting at any time.

Create as copy of	XDXSettingsDialog		×
Name	iiiSettingsDialog		Create
Description	Settings Dialog		Cancel
Framework	Personnel & Payroll (HUMAN RESOURCES)	-	cancer
Group		Ŧ	
Identifier	IIISETTI		

- 5. Press the *Create* button.
- 6. The form is opened for editing. The *Design* tab is selected by default and the design layout of form is shown.
- 7. Display the *Source* by clicking on the tab.



The Source tab shows the source code of the form.

Note: Do not compile this form. We will simply use this code as an example to demonstrate editor features. Leave form iiiSettingsDialog open in the editor.

Step 2. Create a Process and a Function

In this step you will create a new Process iiiPRO01. Processes are containers used to group together a set of functions. When executed, processes appear as a menu. Processes are also used to control characteristics of the functions that they contain.

After creating the process, you will create a function belonging to the process. Functions are programs that contain RDML code. You will use this sample function, along with the sample form, to learn the basics of controlling the display of LANSA code in the editor. The actual meaning of the code is not important.

1. First, create a new process from *New* in the *File* menu, and selecting *Process*:



- a. Enter a *Process name* of **iiiEditorTesting** (where iii=your initials) or DEMPRO01 if you are using a trial version of Visual LANSA.
- b. Enter a *Description* of **Demo Process**.
- c. Uncheck the Open in editor option.

🧈 New Process		×
Name	iiiEditorTesting	Create
Description	Demo Process	Cancel
Identifier	IIIEDITORT	Cancer
		🔲 Open in editor

d. Press the *Create* button.

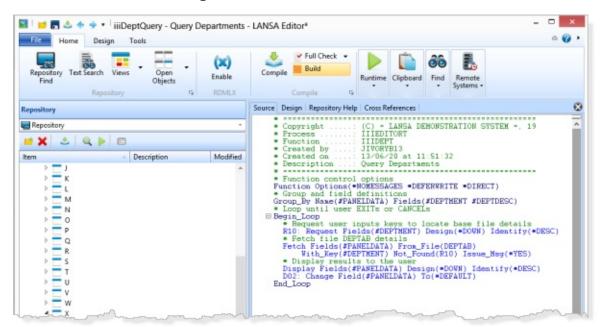
- 2. Use *File* menu / *New* to create a new function:
 - a. Enter a *Process name* of **iiieditortesting** (where iii=your initials) or DEMPRO01 if you are using a trial version of Visual LANSA . Note that this name will be completed as you type, if a match is found in the Repository.
 - b. Enter a *Name* of **iiiDeptQuery** (where iii=your initials).
 - c. Enter a *Description* of **Query Departments.**
 - d. Select *Template* **FRENQ01 Flat Screen Enquiry**.
 - e. **Do not select** *Enabled for RDMLX*.
 - f.. *Open in editor* will be automatically selected.

	New Function	×
Process Name	iiieditortesting	Create
Description	Demo Process	Cancel
Name	iiiDeptQuery	Cancer
Description	Query Departments	Open in Editor
Template	FRENQ01 - Flat Screen Enquiry *	
Identifier	IIIDEPT	
Enabled For RDMLX		

- g. Click the *Create* button.
- h. Complete each *Wizard/Template Prompt* as shown and click *Next*.

Prompt	Enter
Enter the name of the base file to be used by this template	DEPTAB
Do you want this function to be part of an ACTION- BAR style process	Ν
Fields to appear on Display	DEPTMENT
	DEPTDESC
Design fields on the enquiry panel DOWN or ACROSS the screen	DOWN

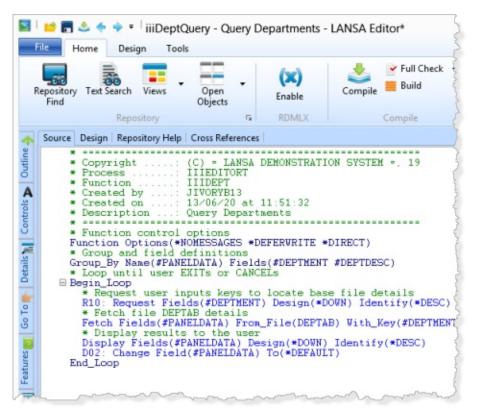
- i. Click *Finish* to create the RDML code.
- 5. The *Source* tab showing the RDML code for this function will be shown.



Step 3. Turn on Autohide

1. Use the context menu on the *Repository* title bar (or whichever tab is showing on the left) to *Autohide* the left hand panels.

The left tab folders are now hidden, leaving you more space to work with the source code in the editor.



Remember, the tabs can be displayed by clicking on them. When the focus leaves the tabs, they automatically hide again.

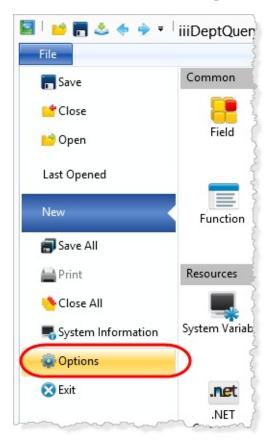
To pin the tabs open again, click on the *Attach* \pm button while the tab is being displayed.

Step 4. Change Formatting Options

In this step you will change how source code is formatted. You can control what lines of code are displayed and how they are displayed.

Separate settings are used to control statements in functions and components (forms, reusable parts, WAMs, etc.).

1. On the *File* menu, click on *Options*. The *LANSA Settings* dialog is displayed. This dialog will be different for Slave and Independent systems.





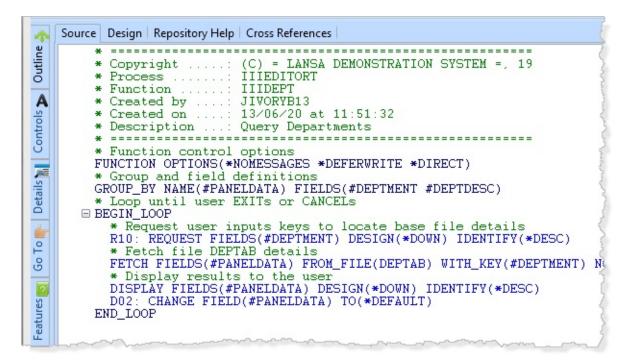
- 2. Select the *Source* icon to view the code formatting options.
 - a. Expand the *Function Formatting* options.
 - b. Select *Upper case* so that commands and keywords are in shown in uppercase.



3. Press the *Apply* button to make the changes to the source code. Click *Yes* in the dialog warning that undo and redo buffers will be lost:

	LANSA Editor	
?	The new formatting options will cause the loss of all buffers for function IIIDEPT. Do you wish to continue	undo and redo
-	barrels for function moet fr. bo you wish to continue	

4. The formatting changes for your function are now visible in the *Source* tab.



- 5. Use the *Options / Source* options to change the formatting of functions back so that the commands and keywords are in *Proper Case*..
- 6. Press the *Apply* button to make the changes to the source code.

Step 5. Editor Source Settings

In this step you will change the line numbering and indentation options in the *Source* view. You will view the form's source code and see how to control the component definitions.

1. Switch to the iiiSettingsDialog form in the editor by using the *Previous* or *Next* buttons on the toolbar.



You can also do this 'switch' using the *Open Objects* menu or the *Outline* tab.

Notice that the source code is showing the details for all component definitions:

4	Design Source Repository Details Repository Help Cross References
Outline	Function Options(*DIRECT) Begin_Com Role(*EXTENDS #PRIM_FORM) Bordericons(SystemMenu) Caption('Settings') Clientheight(201) Clientwid
Details	Dialog providing options for the application appearance Define_Com Class(#FRIM_LABL) Name(#LABL_1) Caption(#MIXIXX043) Displayposition(1) Ellipses(Word) Height(2) Define_Com Class(#FRIM_CAEI) Name(#Fonts) Combohowstyle(DropDownList) Componentversion(1) Displayposition(1) Define_Com Class(#FRIM_CAEI) Name(#CBCL_1) Displayposition(1) Parent(#Fonts) Source(#XDXCaption) Usepickli Define_Com Class(#FRIM_CBCL) Name(#CBCL_2) Parent(#Fonts) Source(#XDXAlpha) Usepicklist(False) Visible(Fal Define_Com Class(#FRIM_LABL) Name(#IExtColor) Caption(*MIXIX0044) Displayposition(3) Ellipses(Word) Height(20) Left(120) Parent Define_Com Class(#FRIM_LABL) Name(#TextColor) Displayposition(4) Ellipses(Word) Height(20) Left(120) Parent
Repository	□Define_Com Class(#PRIM_LABL) Name(#LABL_2) Caption(*MTKTXDX045) Displayposition(5) Ellipses(Word) Height(1) Define_Com Class(#PRIM_SPDT) Name(#TopRight) Displayposition(6) Height(20) Left(120) Parent(#COM_OWNER) Shc Define_Com Class(#PRIM_SPDT) Name(#TopRight) Displayposition(7) Height(20) Left(176) Parent(#COM_OWNER) Shc Define_Com Class(#PRIM_SPDT) Name(#BottonLeft) Displayposition(8) Height(20) Left(176) Parent(#COM_OWNER) Define_Com Class(#PRIM_SPDT) Name(#BottonRight) Displayposition(8) Height(20) Left(176) Parent(#COM_OWNER)
Favorites 🙀	Define_Com Class(#prim_vs.Style) Name(#TextColorStyle) Normbackcolor(Black)
Favo	Define_Com Class(#prim_appl.ICommonDialogColor) Name(#ColorPicker) Reference(#dynamic)
	□Define_Com Class(#PRIM_PHEN) Name(#Apply) Caption(*NTXTXDX047) Displayposition(10) Left(256) Parent(#COM_C Define_Com Class(#PRIM_CMEX) Name(#Transition) Comboboxstyle(DropDownList) Componentversion(1) Displayposi Define_Com Class(#PRIM_CDECL) Name(#CBCL_3) Displayposition(1) Parent(#Transition) Source(#XDXAption) User Define_Com Class(#PRIM_CBCL) Name(#CBCL_4) Parent(#Transition) Source(#XDXAption) User Define_Com Class(#PRIM_CBCL) Name(#CBCL_4) Caption(*HTXTXDX046) Displayposition(12) Ellipsee(Vord) Height(

- 2. Using the *File* menu, select *Options* to open the *LANSA Settings* dialog.
- 3. Select the *Source* icon to view the source code options.



a. Select the *Indentation* and *Line Numbers* options.

- b. Deselect the *Component Definitions* option.
- b. Press Apply.
- 4. You will see that the source code is now showing indent lines and line numbers.

Note that the component definitions (Define_Com) are now compressed so that only the first definition of each set is shown.

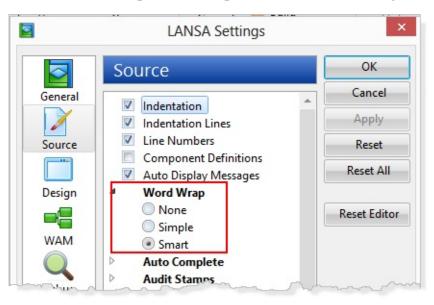
-	Design	Source Repository Details Repository Help Cross References
Outline	00001 00002	Function Options(*DIRECT) -Begin_Com Role(*EXTENDS #PRIM_FORM) Bordericon Clientwidth(346) Formstyle(StayOnTopChild) Top(244) Width(362)
Controls 🖌	00003 00004 00005	★ Dialog providing options for the application Define_Com Class(#PRIM_LABL) Name(#LABL_1) C Height(20) Left(8) Parent(#COM_OWNER) Ta
Details 📶 🛛 C	00011 00012	Verticalalignment(Center) Width(105)
To P	00017 00018 00019 00020	Define_Com Class(#prim_vs.Style) Name(#Text Define_Com Class(#prim_appl.ICommonDialogCol
e	00021	perinc_com crass(*prim_appr.resmonbraiogeor

5. Reset the *Editor Source Settings* to remove the *Line numbers* and *Indentation Lines* and to show all *Component Definition* lines.

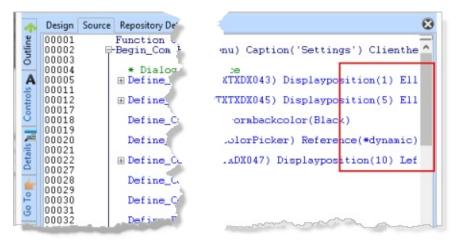
Step 6. Word Wrap

In this step you will change the word wrapping options in the Source view. Word wrapping is used to control how single lines of code are displayed in the editor. Wrapping will ensure that the complete line of code is viewable in the current width of the Source tab.

1. On the File menu select Options, to open the LANSA Settings dialog.



- a. Select the *Source* icon to view the source code options.
- b. Expand the Word Wrap options.
- c. Select the *None* option. The default (shipped) setting is *Smart* word wrap.
- d. Click *Apply* and then *OK*.
- 2. Each line of source code is now shown as a single line.



Optional: You can try the other Word Wrap settings to see how they impact the code display.

- 4.Open the *Settings* dialog again and reset the *Word Wrap* setting to *Smart*.
- 5. Close the iiiSettingsDialog form in the editor.



Close the form using the blue cross at the top right of the editor.

No changes should have been made to the code. (If you have made a change by mistake, simply press No when asked to save changes.)

6. The editor will display the iiiDeptQuery function.

Step 7. Submit a Compile

In this step you will compile the iiiDeptQuery function and view the compile messages.

1. One the *Home* ribbon, click on the highlighted button in the corner of the *Compile* options area.

Home Design Tools		
ry Text Search Views	Open Objects 5 RDMLX	Compile
Design Repository Help	Cross References	
* Copyright	(C) = LANSA DEMONSTRATION ILLEDITORT	ON SYSTEM =, 19
* Function	۵	Compile options ×
* Created on		Process/Function Compile Options
* Function control Function Options(*N * Group and field d Group_By Name(#PANE * Loop until user E Begin_Loop * Request user in R10: Request Field * Fetch fields(#PAN * Display results Display Fields(#P D02: Change Field End_Loop	Processes and Functions (1)	Compile process only if necessary Compile all process functions Compile functions only if necessary Keep generated source Debug enabled Generate HTML Validate numerics Generate XML
1	Use Default Settings	OK Cancel
h	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

The *Compile Options* dialog will be shown.

2. You do not need to Generate HTML or XML. Remove these options. Removing the three compile options as shown, means "compile just this function", regardless of whether VL is aware it has changed. Visual LANSA will now always compile selected functions and the process.

Your settings should appear as follows:

ځ (Compile options ×
Processes and Functions (1)	Process/Function Compile Options Compile process only if necessary Compile all process functions Compile functions only if necessary Keep generated source Debug enabled Generate HTML Validate numerics Generate XML
Use Default Settings	OK Cancel

- 3. Press the *OK* button to submit the compile.
- 4. Resize the bottom area of the editor and display the *Compile* tab.

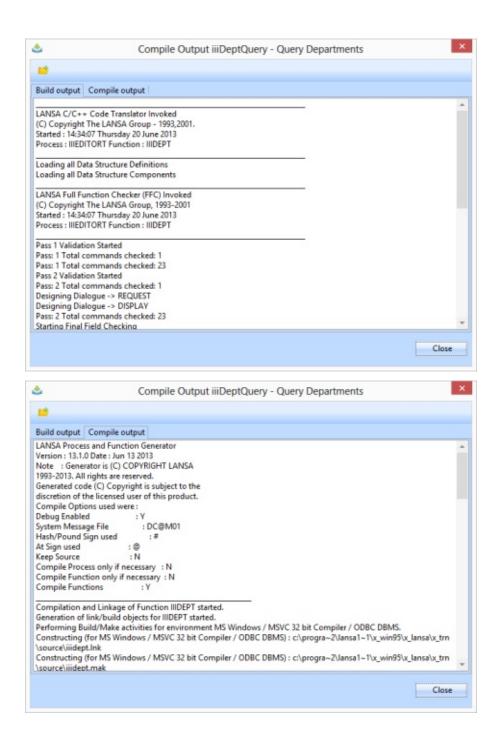
The status of the compile will be displayed in the *Compile* tab. You will see the messages change as the compile is in progress and then completes.

Features 👩 🌀	¥	l I Enc PR	* Display results Display Fields(#F D02: Change Field d_Loop C0010/FFC Complet	ELDALA, Frôm_Frie(DELVAD) to the user ANELDATA) Design(*DOWN) J (#PANELDATA) To(*DEFAULT) ed : +0 warning messages ed : +0 fatal messages is	dentify(*DESC)
-	<				~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Repository 🚛	×	P	Job Status	Description	Results
Repo	7	2	Completed	Compile 2 objects	Compiled 2 of 2
+					
rites	•	×	4) 1
Favorites		Assist	ant 🙏 Compile 🕜 I	Help	
				Ln 22, Col 1 LANSA13SP1	FRN JIVORYB13 TRAINING 🖡

5. Expand the "Completed, Compiled 2 Objects" message:

Select the **iiiDeptQuery** object and click the *set the poly* joblog icon to see the compile messages.

The *Compile Output* dialog will be displayed. You can view the source code build messages as well as the compile messages.



Notes:

a.The logs contain two tabs, with details of the build and the compile.

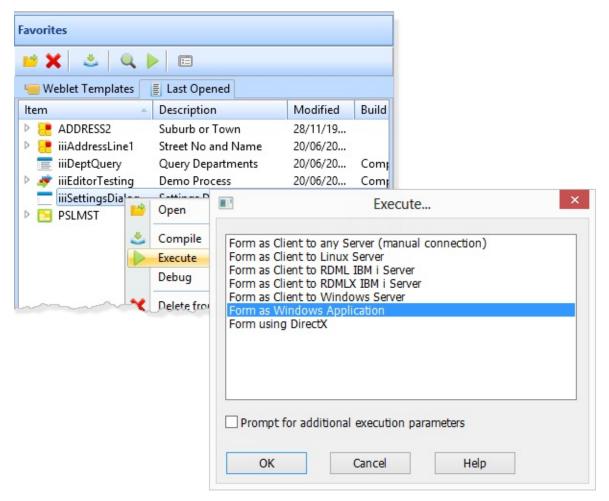
b.The log can be opened in Notepad, in case you need to save the compile logs to send to LANSA Support.

- 6. Close the *Compile Output* dialog.
- 7. Close the iiiDeptQuery function.

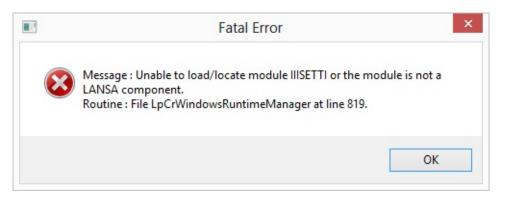
Step 8. Display Error Log

In this step you will cause a fatal error to occur and the view the error log. The error log is associated with run-time errors.

1. On the *Favorites / Last Opened* tab, select *iiiSettingsDialog* and use the context menu to *Execute* the form. In the *Execute* dialog, select *Form as Windows Application*.



2. A fatal error is displayed.



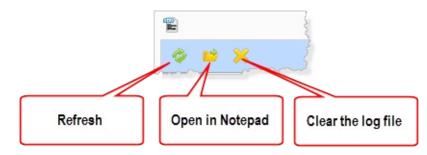
- 3. Close the *Fatal Error* message box (Click *OK*) and *Cancel* the *Message* window.
- 4. On the *Home* ribbon, open the *Error Logs* menu to select *Local*.



to view the Visual LANSA Error Log:

Ø 🖻 🗙					
Details Thursday 20 June 2013 14:50:22 14:49:28 08:49:39 08:47:37 08:45:56 Thursday 13 June 2013 Tuesday 04 June 2013 Thursday 30 May 2013 Tuesday 21 May 2013 Tuesday 21 May 2013	Relea Mess com Rout Job N DBU DBU LOCI DRIV	un 20 14:50:22 2013 se 13.1.0 Build 4075 Windows 8 P age : Unable to load/locate mode onent. ne : File LpCrWindowsRuntimeM lumber: 011292 OS User : John =MSSQLS FORM=IIISETTI PART= =USERID DBII=LX5JOHNI DBIT=N C=YES TASK=TRAINING DEVE=Y I =C:\PROGRA~2\LANSA1~1\X_W =20000 ITRL=4 ITRC=ALL ITHP=)	ule IIISETTI or the mod lanager at line 819. = TRN LANG=ENG USE MSSQLS GUSR=QPGM DATF=DMY IN95 PRTR=LPT1 DBU	ER=JIVORYE IR HLPC=1 IG=N ITRO:	813 509852

The toolbar buttons enable the error log to be refreshed, opened in Notepad or cleared.



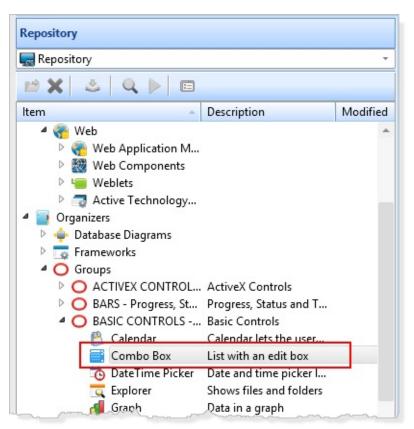
This error occurs simply because the *iiiSettingsDialog* form is not compiled.

5. Close the *Error log* window.

Step 9. Display Feature Help Text

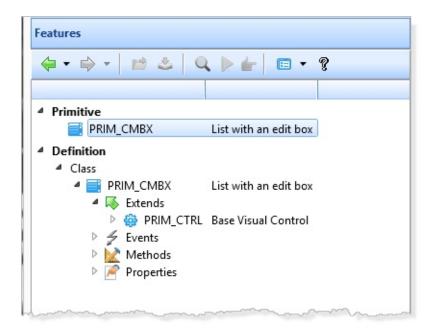
In this step you will use the help text tab to view feature help for components. Feature help is used to display detailed information about components. Feature help is particularly important as you edit forms and reusable parts.

- 1. On the *Repository* tab, under *Organizers*, expand *Groups* and then expand *Basic Controls*.
- 2. Select the Combo box control.



3. Press **F2**.

The *Features* tab will be displayed, showing the combo box control's *Properties, Events* and *Methods*.



Note that this tab shows the *Feature help (F2)*. As you learned in the *Visual LANSA User Interface Tutorials*, there is other context-sensitive help available when you press F1. F1 will always link you to a suitable reference in the online guides.

4.Expand *Methods* and double click on any of the component's methods. Help text is displayed explaining how the selected method can be used.

Features	
⇔ • ⇒ • ≅ ≤ Q, ≥ <u>+</u> Ξ • ?	
Primitive PRIM_CMBX List with an edit box	
Definition Class	
 PRIM_CMBX List with an edit box Ketends 	
PRIM_CTRL Base Visual Control Sector Sect	
 Methods 	
CloseDropDown Result Boolean of (False, True)	
 Fadeln FadeOut MoveFrom MoveTo OpenDropDown Realize Scale SelectText 	
SetFocus SetValueAt ShowMessages Unrealize UpdateDisplay Properties	 CloseDropDown method The CloseDropDown method forces the list box portion of a DropDown or DropDownList style combo box to be closed. CloseDropDown will return True if the control has a list box to drop down and that list box is already down; otherwise, False.
	Assistant Compile W Help ISA13SP1 TRN JIVORYB13 TRAINING ENG Audit Off LANSA XHTML DirectX •

Summary

Important Observations

- You can customize the way source code is displayed in the editor.
- You can toggle the display of line numbers and indentation on or off.
- You can hide DEFINE_COM statements.
- The Help Text tab shows Feature help for selected components.
- The Compile tab shows the compile status of objects.

What I Should Know

- How to open process, functions and components in the editor.
- How to customize the display of source code in the editor.
- How to show and hide line numbers and indentation of the source code.
- How to change the way statements are formatted.
- How to hide DEFINE_COM statements in the source code.
- How to use feature help for components.
- How to view compile status messages.

VED020 - Edit Source Code

Objectives:

- To learn how to use keyboard shortcuts to navigate within the code in the Source tab.
- To learn how to use keyboard shortcuts to comment and uncomment code.
- To learn how to use copy, paste, and undo features.
- To learn how to find and replace text in the Source tab when editing a function or a component.

The focus of this exercise is learning how to navigate within the editor and use some basic editor features. The purpose of the programs and the actual meaning of the RDML commands is not important. The RDML Programming tutorials teach the basic coding practices.

To achieve these objectives you will complete the following:

- Step 1. Cursor Position
- Step 2. Position the Current Line
- Step 3. Comment Lines
- Step 4. Copy and Paste
- Step 5. Find Text
- Step 6. Use the Toolbar Find Button
- Step 7. Use the Find Dialog
- Step 8. Find and Replace
- Step 9. Search Text in Several Objects
- Summary

Before You Begin:

You may wish to review the following topics in the *Visual LANSA User Guide*:

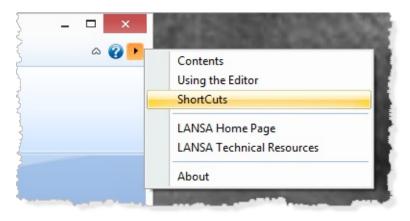
- Keyboard Shortcuts
- Source Tab

You should complete all previous tutorials.

Step 1. Cursor Position

In this step you will use keyboard shortcuts to move the cursor to different positions in the code displayed in the Source view.

1. From the *Help* menu button at the top right of the editor, select *Shortcuts*:



2. The *Help* tab will initially be displayed docked at the bottom of the editor. Float and resize the *Help* tab. Use the links to review some of the many editor shortcut options:

	Help	x
4	Keyboard Shortcuts	~
\$	For most features, you can use Visual LANSA with or without a mouse. The keyboard can also be used to select items and actions within a window. The Screen and Report Designing facilities in Visual LANSA do need a mouse. Keyboard	
-	Move the cursor in the Source tab	
-	Position the current line	
5	Tag and comment lines	
	Collapse and expand code	
2	Format text	
	Deletions	
	Copy, cut and paste	
	Help	
	Selection	
	Undo/Redo	
	Find and replace	
	Command Assistant Keystrokes	
	Menu editor	
	Select with the mouse	
	1 Editor Basics	

- 3. Close the *Help* tab. Its size and position will be remembered.
- 4. Use the *Favorites / Last Opened* tab, to open form **iiiSettingsDialog** in the editor.

5. Display the *Source* tab.

Remember, your source code appearance is determined by the editor display settings. (Refer to exercise VED010 - Format Source Code).

Design	Source Repository Details Repository Help Cross References
00001	Function Options(*DIRECT)
00002	BEGIN COM ROLE(#ÈXTENDS #PRIM FORM) BORDERICONS(SystemMenu) CAPTION('Settings') CLIENTHEIGHT(201) CLIENTWIDTH(346) FORMSTYLE(StayOnThopChild) HEIGHT(239) LEFT(666) STYLE(#XDXStyles(Background)) TOP(366) WIDTH(352)
00003	
00004	* Dialog providing options for the application appearance
00005	□ DEFINE_ČOM CLASS(#PRÎM_LABL) NAME(#LÂBL_1) CAPTIOÑ(*MTXTXDX043) DISPLAYPOSITION(1) ELLIPSES(Word) HEIGHT(20) LEFT(8) PARENT(#COM_OWNER) TABPOSITION(1) TABSTOP(False) TOP(8) VERTICALALIGMNENT(Center) WIDTH(105)
00006	DEFINE_COM_CLASS(#PRIM_CMBX) NAMÉ(#Fonts) COMBOBOXSTYLE(DropDownList) COMPONENTVERSION(1) DISPLAYPOSITION(2) FIXEDHEIGHT(False) LEFT(120) PARENT(#COM_OWNER) SHOWSELECTION(False) SHOWSELECTIONHILIGHT(False) TABPOSITION(2) TOP(8) WIDTH(217)
00007	DEFINE_COM_CLASS(#PRIM_CÉCL) NAME(#CECL_1) DISPLAYPOSITION(1) PARENT(#Fonts) SOURCE(#XDXCaption) USEPICKLIST(False)
00008	DEFINE_COM CLASS(#PRIM_CBCL) NAME(#CBCL_2) PARENT(#Fonts) SOURCE(#XDXAlpha) USEPICKLIST(False) VISIBLE(False)
00009	DEFINE COM CLASS (#PRIM LABL) NAME (#17 extColor) CAPTION (*MTXTXDX044)

- 6. If you have not already done so, turn on *Autohide* to maximize the display area for your source code.
- Move the cursor to the end of the window by pressing the **End** key twice.
 Note: This moves to the end of the window, not the end of the source.
- 8. Move the cursor to the top of the window by pressing the **Home** key twice.
- 9. Move the cursor to the end of the source by pressing the **End** key three times.
- 10. Move the cursor back to the top of the source by pressing **Ctrl + Home**.
- 11. Use the down arrow key to move to the BEGIN_COM statement in the Source tab, and then use the **Ctrl + right arrow key** to move to next part of this BEGIN_COM statement.

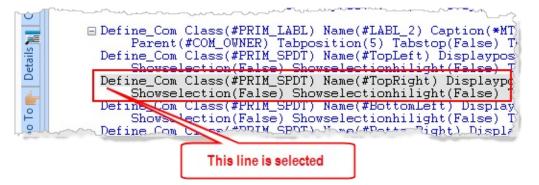


Step 2. Position the Current Line

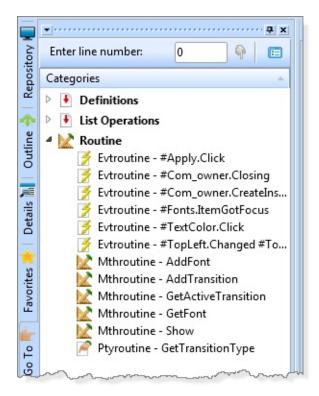
In this step you will position the current line in the editor using shortcut keys and the *Go To* tab.

- 1. Position the cursor on any DEFINE_COM statement towards the bottom of the window..
- 2. Press **Ctrl** + **T** to move the DEFINE_COM statement to the top of the editor window.
- 3. Press **Ctrl + M** to move the DEFINE_COM statement to the middle of the editor window.

Hint: Clicking to the left of any statement will select the whole line, making the above moves easier to see.



4. Display the *Go To* tab and pin it place. Expand the *Routine* node:



5. Double-click on the **Evtroutine - #Apply.click**.

The editor will position at first line of this routine in the source code.

- 6. Press **Ctrl** + **B** to move the event routine line to the bottom of the editor window.
- 7. Press **Ctrl** + **T** to move it to the top of the editor window.

Step 3. Comment Lines

In this step you will learn how to comment and uncomment lines with a keyboard shortcut.

1. Using the cursor, select the highlighted lines of code in the event routine.

Design Source	e Repository Details Repository Help Cross References
00183 00184	E-Evtroutine Handling(#Apply.Click)
00185 00186	Define_Com Class(#prim_vs.Style) Name(#PanelStyle) Define_Com Class(#prim_vs.Style) Name(#ApplicationStyle)
00187 00188	#ActiveTransition := #Com owner.GetActiveTransition
00189 00190	
100191	- I was a ser a se

2. Press **Ctrl** + **W** to change these lines of code to comment lines.

Design Source	e Repository Details Repository Help Cross References
00183	-Evtroutine Handling(#Apply.Click)
00184 00185 00186 00187	<pre>* Define_Com Class(#prim_vs.Style) Name(#PanelStyle) * Define_Com Class(#prim_vs.Style) Name(#ApplicationStyle)</pre>
00188	#ActiveTransition := #Com_owner.GetActiveTransition
00190 00191	E-If (#StyleChanged)
00192 00193 V	* Apply settings to the application #ApplicationStyle.FaceName := #Com_owner.GetFont Name APPLICATIONSTYLE could not be found.
00194 🔻	#ApplicationStyle.TextColor := #TextColorStyle.NormBackColor Name & PPLICATIONSTYLE could not be found.

Notice the editor has immediately highlighted errors due to the missing code. 3.Use Ctrl+Shft+W to change these lines back to RDMLX language commands.

Step 4. Copy and Paste

The editor uses standard Windows shortcut keys for *copy*, *cut and paste* operations.

Use **Ctrl+Home** to move to the top of the source code.

In this step you will learn how to copy and paste text.

- 1. Position the cursor in front of the BEGIN_COM statement.
- 2. Press **Ctrl + Shift + right arrow** to select the text BEGIN_COM.
- 3. Press **Ctrl** + **C** to copy the text.
- 4. Position the cursor before the BEGIN_COM and press *Enter* to add a blank line above the BEGIN_COM statement.
- 5. With the cursor on the blank line press **Ctrl** + **V** to paste in the copied text.
- 6. Select the newly pasted text.
- 7. Press **Ctrl** + **Z** to undo the paste operation.

Step 5. Find Text

In this step you will search text in the Source tab.

1. Press **Ctrl** + **F** to display the *Find* dialog.

You may also use *Find* or *Replace* from the *Find* button on the *Home* ribbon.



2. Type EVTROUTINE into the Find what: field.

3	Find	
Fi <u>n</u> d what:	evtroutine	<u> </u>
□ Match whole word only Direction □ Match case □ Up		<u>T</u> ag All
Wrap Wrap	<u>D</u> own	Cancel

3. Press the *Find Next* button.

You will be positioned to the first EVTROUTINE.

- 4. Press **Esc** to close the *Find* dialog.
- 5. Press **F3** to find the next occurrence of EVTROUTINE.

Step 6. Use the Toolbar Find Button

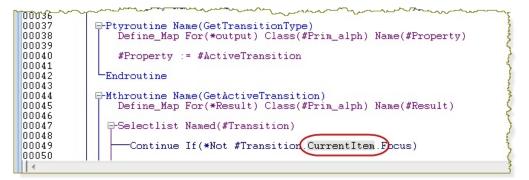
In this step you will find text using the *Find* button on the *Home* ribbon.

- 1. Move the cursor back to the top of the form by pressing **Ctrl + Home**.
- 2. Click on the *Find* button on the *Home* ribbon, position the cursor in the input box and type **currentitem**.



3. Press **Enter** on the keyboard.

The first occurrence of the string is located in the code.



4. Use the *Next* in the *Find* button dialog to find the next occurrence of **currentitem**.



- 5. Press **F3** to find where the next occurrence of the string **Currentitem** is located in the source code.
- 6. Press **Shift** + **F3**. The previous occurrence of the string **currentitem** is located in the source code.

Note: The *Find* dialog stores a list of your search values for your current VL session.

The appearance of the *Find* dialog in the *Home* ribbon, varies depending on how your Editor has been resized, for example:



Step 7. Use the Find Dialog

In this step you will use the *Find* dialog to mark all occurrences of the string **currentitem**.

- 1. Place the cursor on the BEGIN_COM statement at the top of the source code.
- 2. Press **Ctrl** + **F** or use the *Find* button on the *Home* ribbon to display the *Find* dialog.

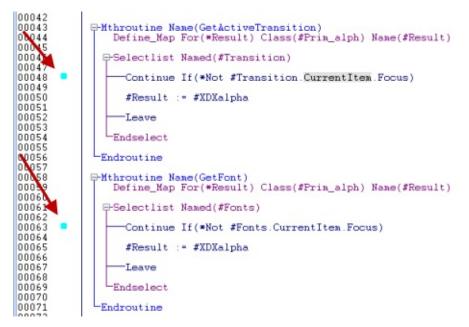
The Find dialog is displayed with BEGIN_COM set as the search string.



3. In the Find what drop-down list select **currentitem**.

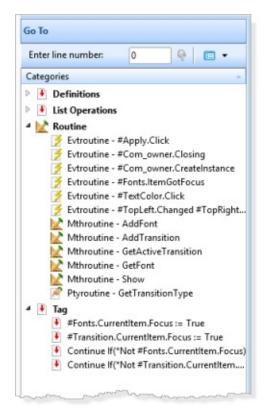
	Find	
Find what:	currentitem	Find Next
Match whole wo	Begin_Com currentitem	Tag All
Match case ✓ Wrap	Down	Cancel

- 4. Click the *Tag All* button to mark all occurrences of the string **currentitem** in the source.
- 5. Scroll down the code. Notice how every occurrence of **currentitem**, is marked with a small blue rectangle.



Tags are temporary and are not saved with the source.

Tags are shown on the *Go To* tab:



6. Click *Cancel* to close the Find dialog box.

Step 8. Find and Replace

In this step you will replace the string **application** with **system**.

- 1. Press **Ctrl** + **H** to bring up the *Replace* dialog.
- 2. Enter **application** in the *Find what* field.
- 3. Type system in the *Replace with* field.

Select the *Match whole word only* check box.

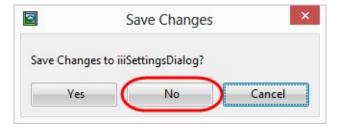
	Replace		
Find what:	application	*	Find Next
Replace with:	system	-	Replace
Match whole w Match case	ord only		Replace All
Wrap			Cancel

4. Click the *Replace All* button.

All occurrences of the string **application** are replaced with **system**.

00084		-Endif
00086		-Endroutine
00088		E-Evtroutine Handling(#Com_owner.CreateInstance)
00090		* Allowed fonts for the system
00091		* Could alos be loaded from Windows font list #Com_owner.AddFont("Segoe UI")
00093		#Com_owner.AddFont("Tahoma") #Com_owner.AddFont("Courier New")
00095		<pre>#Com_owner.AddFont("Arial") #Com_owner.AddFont("Verdana")</pre>
00097		
00098	•	Get_Entry Number(1) From_List(#Fonts) #Fonts.CurrentItem.Focus := True

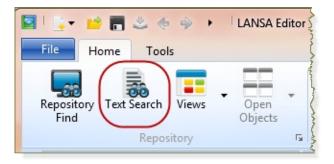
5. Close the iiiSettingsDialog form. Be sure to answer *No* when asked if you want to save the changes.



Step 9. Search Text in Several Objects

In this step you will search for text in all Repository objects that contain source code.

1. Click on the *Text Search* button on the *Home* ribbon.



2. In the *Text Search* dialog specify to search all forms, reusable parts and functions for the text SELECT. Limit the search to objects starting with XDX*. Note the wildcard '*'.

-		Text Search	
Object Types	Search for text	SELECT	Search
Forms	Like name	XDX*	
Reusable parts	Like name	ADA	Cancel
Functions			
🕐 WAMs			
Weblets			
Fields			
E Fields			

3. Click the *Search* button.

The Text Search tab now shows all the forms, reusable parts and functions starting with XDX that contain the text SELECT.

×	ir.	Qualified Object	Description	Line Number	Line of Code
-		 Completed 	30 matches found		Search for SELECT in XDX*
	×	XDX00002	DirectX Examples	52	* Transition animation between the
1	0	XDX00002	DirectX Examples	144	Mthroutine Name(ActivatePanel) H
1	0	XDX00002	DirectX Examples	159	* Hook up the animation to the sele
1		XDX00056	Settings Dialog	6	Define_Com Class(#PRIM_CMBX) N
1		XDX00056	Settings Dialog	13	Define_Com Class(#PRIM_SPDT) Na
1		ZDX00056	Settings Dialog	14	Define_Com Class(#PRIM_SPDT) Na
		- XDX00056	Settings Dialog	15	Define_Com Class(#PRIM_SPDT) Na
÷			comment and	manne	

Summary

Important Observations

- There are various keyboard shortcuts which make working in the Source tab more efficient.
- The quickest way to search through source code is to use the Find feature in the toolbar.
- You can mark all the occurrences of a string using the Find dialog

What I Should Know

- How to use keyboard shortcuts to navigate within the source code.
- How to comment and uncomment code.
- How to go to different positions in the source code using the go to feature of the Outline tab.
- How to use copy and paste code.
- How to undo changes.
- How to find strings in the source code of a component or function.
- How to find and replace strings in the source code.

VED030 - Auto Complete and Command Assistant

Objectives:

- To learn how to use Auto Complete.
- To learn how to use the Command Assistant.
- To display the online help for RDML commands in the LANSA Editor.

The focus of this exercise is to learn how to use the Auto Complete and Command Assistant in the editor. The purpose of the programs and the actual meaning of the RDML commands is not important. The RDML Programming tutorials teach the basic coding practices.

To achieve these objectives you will complete the following:

- Step 1. Display the Command Assistant
- Step 2. Use the Command Assistant
- Step 3. Use Auto Complete Prompter
- Step 4. Use the Online Help Command
- Summary

Before You Begin:

You may wish to review the following topics in the *Visual Guide User Guide*:

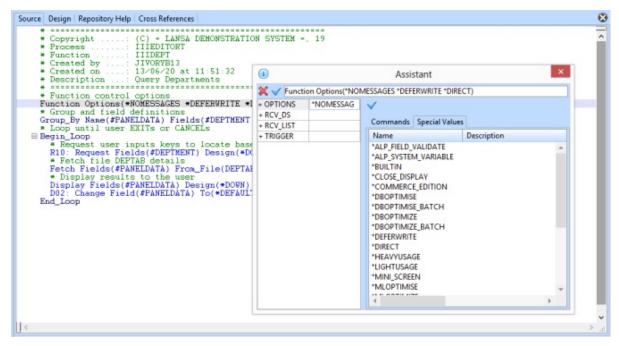
- Auto Complete
- Command Assistant

In order to complete this tutorial, you must have completed the previous steps in this tutorial.

Step 1. Display the Command Assistant

In this step you will learn how to use the *Command Assistant* when editing code. The *Command Assistant* is a prompting facility that helps you to build RDML commands.

- 1. Use *Favorites* tab to open the **iiiDeptQuery** function in the editor's *Source* tab.
- 2. Position the cursor on the **FUNCTION** command. Press **F4** to open the *Command Assistant* tab. It will initially be shown docked, at the bottom of the editor.
- 3. Float the *Command Assistant* tab and resize it. It is usually a good idea to use the *Command Assistant* in this way. When you close it, the settings will be remembered and used the next time it is opened.



- 4. Expand the OPTIONS parameter of the FUNCTION command.
- 5. Select the *Special Values* tab. You will see a list of all options than can be selected and values can be added or changed.

i	Assistant			×	
💥 🧹 Function	Options(*NOMESS	SAGES *DEFERWR	RITE *DIRECT)		
- OPTIONS	*NOMESSAGES	\checkmark			
Special function	*NOMESSAGES	1			
Special function	*DEFERWRITE	Commands	Special Values		
Special function	*DIRECT	Name		Description	
Special function		*ALP_FIELD_	VALIDATE		
+ RCV_DS		*ALP_SYSTEM	VARIABLE		
+ RCV_LIST		*BUILTIN			
+ TRIGGER		*CLOSE_DISP			
	·	*COMMERCE	-	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	m

6. Close the *Command Assistant*.

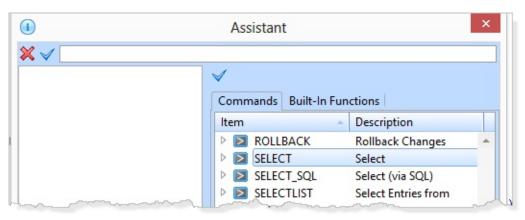
Step 2. Use the Command Assistant

In this step you will learn how to use the *Command Assistant* to create SELECT database command to retrieve all fields from the PSLMST file. The finished command will appear as follows:

```
SELECT FIELDS( #ADDRESS1 #ADDRESS2 #ADDRESS3 #DEPTMENT #EMPNO )
FROM_FILE(PSLMST)
ENDSELECT
```

Note: Experienced developers could quickly type this command into the editor, and the *Autocomplete* assistant would complete code and variables as they are typed (see later). The *Command Assistant* is intended to help new developers build this command.

- 1. Insert a blank line after the BEGIN_LOOP command by positioning the cursor at the end of the line and pressing *Enter*.
- 2. Press **F4** to re-display the *Command Assistant* as a floating window. Note that it has the size and position which you gave it earlier.
- 3. Select the *Commands* tab.
 - a. Locate the SELECT command. (You can simply set focus on an item in the list of commands and type the letter S to position to commands starting with S.)



- b. Press *Enter* to select the SELECT command in the list of commands.
- c. The SELECT command is displayed in the *Command Assistant*.
- 4. Use the *down cursor key* to move to the FROM_FILE parameter.
 - a. Select the *Files* tab.
 - b. Use the *Tab* key to move to the *File name* field. Type PS in the File name

field to display files starting with PS.

(i)		Assistant	×
💥 🧹 Select Fie + FIELDS	elds() From_File()		
+ FROM_FILE WHERE + WITH_KEY NBR_KEYS GENERIC	*WITHKEY *NO	Commands Files Result value filters File name File des ps	cription
IO_STATUS	*STATUS	Item	Description
IO_ERROR VAL_ERROR END_FILE ISSUE_MSG	*ABORT *LASTDIS *NEXT *NO		PSLEVENT by event da PSLEVENT by type, dat Personnel Images Personnel
LOCK RETURN_RRN + OPTIONS	*NO *NONE	 PSLMST PSLMST1 PSLMST2 	Personnel by Deptme Personnel by Surname

- c. Select the **PSLMST** file and press Enter.
- d. The *From_File* parameter in the SELECT command will be updated.
- 5. Click in the FIELDS parameter in the Assistant

Click on the *Fields by File* tab and expand the file PSLMST to display all fields in the PSLMST file.

Select Fir	Ide0 Erem Elle/P	CI MCT)	
Select Fie Select Fie FIELDS FROM_FILE WHERE WITH_KEY NBR_KEYS GENERIC IO_ERROR IO_ERROR END_FILE ISSUE_MSG LOCK RETURN_RRN OPTIONS	elds() From_File(P PSLMST "WITHKEY "NO "STATUS "ABORT "LASTDIS "NEXT "NO "NO "NO "NONE	SLMST) Commands Variables Fields by File Reposito Result value filters File name File description ps Item Description PSLMST Personnel by Deptme PSLMST2 Personnel by Surname PSLMST2 Personnel by Surname EMPN0 Employee Number SURNAME Employee Surname GIVENAME Employee Surname GIVENAME Employee Surname ADDRESS1 Street No and Name ADDRESS2 Suburb or Town ADDRESS3 State and Country POSTCODE Post / Zip Code PHONEHME Home Phone Numbur STARTDTER Start date (YY)	
		DEPTMENT Department Code SECTION Section Code SALARY Employee Salary STARTDTE Start Date (DDMMYY) TERMDATE Termination Date (DD MNTHSAL Monthly Salary P PSLMST1 Personnel by Deptme	•

You can now choose the fields that you want to include in the SELECT statement. For this example, simply select the EMPNO, ADDRESS1, ADDRESS2, ADDRESS3 and DEPTMENT fields. Use the Ctrl+left mouse button to select the fields required.

Select Fie Select Fie FIELDS FROM_FILE WHERE WITH_KEY NBR_KEYS GENERIC IO_STATUS IO_ERROR VAL_ERROR END_FILE ISSUE_MSG LOCK	*WITHKEY *NO *STATUS *ABORT *LASTDIS *NEXT *NO	Assistant	
RETURN_RRN + OPTIONS	*NONE	SURNAME Employee Surname GIVENAME Employee Given Nam ADDRESS1 Street No and Name ADDRESS2 Suburb or Town ADDRESS3 State and Country POSTCODE Post / Zip Code PHONEHME Home Phone Number PHONEBUS Business Phone Numb STARTDTER Start date (YYMMDD) TERMDATER Termination Date (YY DEPTMENT Department Code SECTION Section Code	

6. Press *Enter*. The *Fields* parameter in the SELECT command will now be updated:

(i)	Assistant					
Select Fields(#EMPNO #ADDRESS1 #ADDRESS2 #ADDRESS3 #DEPTMENT) From_File(PSLMST)						
+ FIELDS	#EMPNO #ADDRE	\checkmark				
+ FROM_FILE	PSLMST					
WHERE		Commands Variables Fields by File Repository Fields				
+ WITH_KEY		New qualification #ADDRESS1				

7. Close the Command Assistant. Your code now looks like the following:

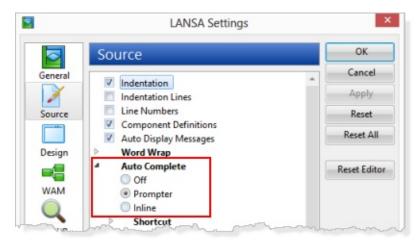
Select Fields(#EMPNO #ADDRESS1 #ADDRESS2 #ADDRESS3 #DEPTMENT)
 From_File(PSLMST)
 Endselect

Step 3. Use Auto Complete Prompter

In this step you will learn how to use the *Auto Complete* prompter when entering a SELECT command to retrieve all fields from the PSLMST file. You will see how *Auto Complete* will show you all parameters for commands. You will begin by deleting the SELECT statement created with the *Command Assistant* and you will code the following commands:

ESELECT FIELDS(*ALL) FROM_FILE(PSLMST)

1. From the *File* menu select *Options*. Select the *Source* options. Make sure that *Auto Complete* has been set to *Prompter*.



- 2. Click the *OK* button to close the *LANSA Settings* dialog.
- 3. Select the SELECT and ENDSELECT statements and press the *Delete* key. You should have a blank line in the editor.
- 4. Type S on the blank line. *Auto Complete* shows a list of commands starting with S:

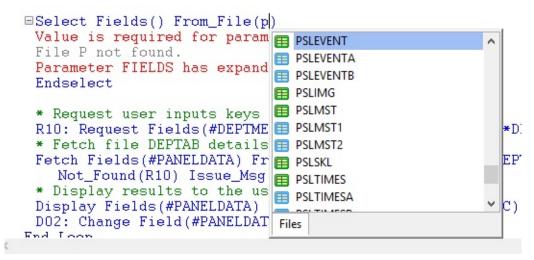
BEGIN	LOOP	DIII 10 01	
V s			
s 🔁	SELECT		nd.
	SELECT_SQL		
* 🖻	SELECTLIST		0.10
R 🗖	SET		$\tilde{\mathbf{T}}$
* ≥	SET_ERROR		
F ව	SET_MODE		M_FI
* 国	SET_REF		r
D 🔊	SIGNAL		ESIG
D 🔁	SKIP) TC
ENI 🔁	SORT_LIST		-
Ca	ommands		
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		~~~

5. Press enter to select the SELECT command.

The *Auto Complete Prompter* adds the SELECT command with its mandatory parameters to the line. The ENDSELECT command is also added.

6. Type P in the *From_File* parameter.

Auto Complete displays a list of files starting with P.



- 7. Select the **PSLMST** file and press *Enter*. The FROM_FILE parameter will be completed.
- 8. Use the left arrow key to move to the *Fields* parameter. Press the spacebar. The *Auto Complete* drop-down will show the fields from the PSLMST file.

⊟Select Fields(	) From_File(PSLMST)		
Value is requir	ADDRESS1	^	
Parameter FIELD Endselect	ADDRESS2		fi
LIUSETECC	ADDRESS3		
* Request user	S DEPTMENT		fi
R10: Request Fi	S EMPNO		N)
* Fetch file DE	GIVENAME		
Fetch Fields(#P	MNTHSAL		M
Not_Found (R1)	PHONEBUS		
* Display resul Display Fields(	PHONEHME		de
D02: Change Fie	POSTCODE	¥	-
End_Loop	Fields in File Groups and Lists Field <	>	

9. Use the *Tab* key to select the *Special Values* tab.

*ALL *ALL_REAL	
*ALL_VIRT	
*EXCLUDING	
*INCLUDING	

10. Select *ALL and press Enter.

The SELECT command definition is complete.

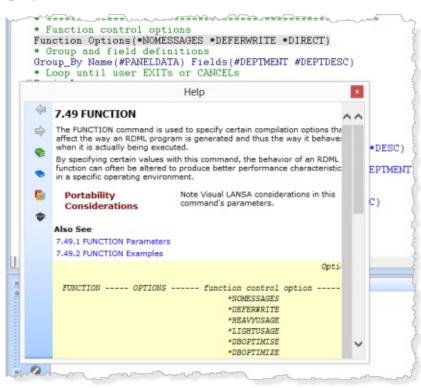
You may want to try to use the *AutoComplete* setting Inline which completes your code on the same line as you type. When you are learning RDML, the recommended setting for *AutoComplete* is Prompter. You may find the *Inline* option faster once you develop some RDML programming skills.

You can turn off *AutoComplet*e and then access it when required by using the *Ctrl+Space* keys anywhere on a command line.

### Step 4. Use the Online Help Command

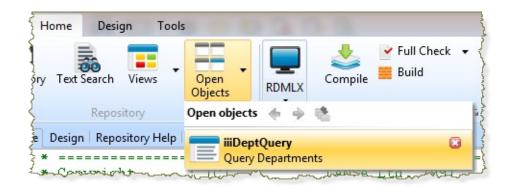
In this step you will review the online help available for RDML commands in the LANSA Editor's *Source* tab.

- 1. Set focus anywhere on the FUNCTION command and press F1 to display the online help.
- 2. The help for the FUNCTION command from the *Technical Reference Guide* will be displayed.



**Note:** Help will be displayed in a floating window, because this was your last setting when using the *F4 Command Assistant*. The size and position was remembered when you closed it.

- 3. Try setting focus to other commands in the function and press F1 to review the online help.
- 4. Close the function. Do not save any changes. You can close any open object from the *Open Objects* button:



#### Summary

#### **Important Observations**

- You can use Auto Complete to quickly enter commands as you type them in the editor.
- You can use Command Assistant to help you structure, review and enter commands.

# **Tips & Techniques**

- The Command Assistant is a very efficient means of building code. As a new developer it is very helpful as it lists all parameters that apply to a command.
- You can press F4 to display the command assistant for an existing statement in order to prompt the command and change its values.
- The Auto Complete feature is sensitive to the cursor position in the command as well as the text that has been entered in the command line. As you move the cursor over the command, you will see the options listed by Auto Complete change.

# What I Should Know

- How to use Command Assistant.
- How to use Auto Complete.
- How to display online help for RDML commands in the LANSA Editor.

### **VED040 - Execute Applications**

#### **Objectives:**

- To show how applications can be executed from within Visual LANSA or from the desktop.
- To learn how to execute processes. These applications are designed for the *Universal Interface* which means they will execute on the IBM i, or under Windows or with a very simple Web interface.
- To learn how to execute forms.

To achieve these objectives you will complete the following:

- Step 1. Determine Compile Status
- Step 2. Execute a Form
- Step 3. Review the XDXExamples Application
- Step 4. Execute the Application from the Execution History List
- Step 5. Execute the Form from the Windows Start Menu
- Summary

## **Before You Begin:**

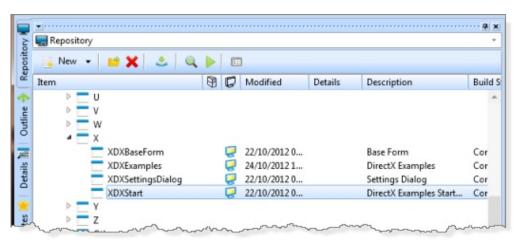
You may wish to review *Executing Applications* in the *Visual LANSA User Guide*:

In order to complete this tutorial, you must have completed the previous tutorials.

#### **Step 1. Determine Compile Status**

In this step you will confirm the compile status of form XDXStart.

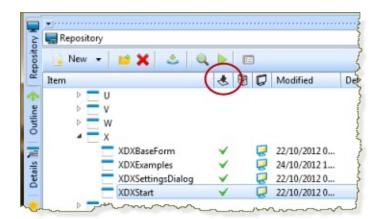
1. Using the *Repository* tab, locate form **XDXStart**. Notice that the *Repository* tab has several columns.



2. Right-click on any column heading to display a menu showing which columns are shown in the tab.

- N	• Repository						
Repositon	🔋 New 🔻 📫 🗶 🔍	►	E	)			
å	Item	8	Ø	Mod	find	Details Description	-
4	▶ <b>U</b>				~	Item	
	> - v				~	Description	
Outline	⊳ <b>—</b> w				~	Modified	
Ő	4 🗖 X				~	Build Status	
-	XDXBaseForm			22/10		Details	
Details 🕅	XDXExamples		ĕ	24/10	-	Column Name	15
etai	XDXSettingsDialog		ĕ	22/10			
•	XDXStart			22/10	-	Туре	s Start.
*	Þ Y		-		~	Task	1
ΰ	Þ 🗖 Z					PC Name	
-ti	D Other					Identifier	
Favorites	Functions				-	Debug Enabled	
-	Processes			(		Local Compile State	
	Resources					Local DLL State	
-		~~	~~			~ Bour ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	when

3. Make sure that the *Local Compile State* column is displayed.



If a column only has an icon as a heading, place the cursor over the column heading to see its tool tip. Remember that you can also rearrange and hide columns in the *Repository* tab. (Refer to *Information about Objects* in the *Visual LANSA User Guide* for how to rearrange columns.)

4. Check that the form XDXStart is compiled. A green tick in the *Local Compile State* column indicates the component is compiled. If necessary, select the form, right-click and choose the *Compile* option from the context menu to submit a compile of the form.

## Step 2. Execute a Form

In this step you will execute the form XDXStart form from the development environment.

- 1. On the *Repository* tab, right-click on XDXStart to display the associated context menu.
- 2. Select the *Execute* option. (You can also use *Ctrl+Shift+E*.)

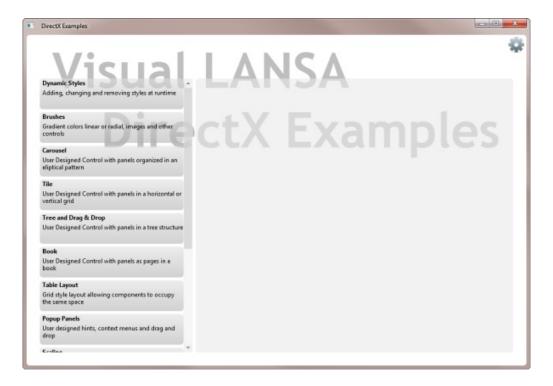
The *Execute*... dialog is displayed:

Form as Cli	ent to any Server (manual co ent to Linux Server ent to RDML IBM i Server	onnection)
Form as Cli	ent to RDMLX IBM i Server	
	ent to Windows Server	
	ndows Application DirectX	
Form as W Form using		

3. Select the Form using DirectX option.

Make sure the *Prompt for additional execution parameters* option is not selected.

- 4. Click the *OK* button.
- 5. When Form XDXStart is executed, an animation component is used to display a splash screen, while the main form XDXExamples is loading:



## Step 3. Review the XDXExamples Application

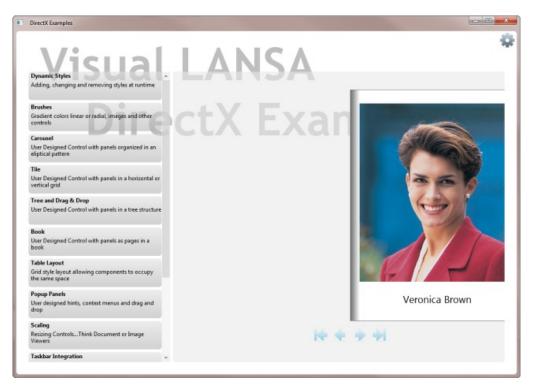
In this step, you will review a very small part of the *DirectX Examples* application. At a later stage you should review the techniques demonstrated, which you may want to include in your own company's modern Windows applications.

1. A set of panels appear in a list on the left side of the form, and serve as a menu into the different parts of the DirectX application. The application is all about demonstrating various interface designs and techniques which can be used in the DirectX interface.

Click on the Book panel:

	DirectX Examples
	Visual
	Dynamic Styles
	Adding, changing and removing styles at runtime
	Brushes
	Gradient colors linear or radial, images and other controls
	Carousel
	User Designed Control with panels organized in an eliptical pattern
	Tile
	User Designed Control with panels in a horizontal or vertical grid
	Tree and Drag & Drop
	User Designed Control with panels in a tree structure
/	Book
	User Designed Control with panels as pages in a book
1	Table Layout a more

Employee images are displayed in the form of a book.

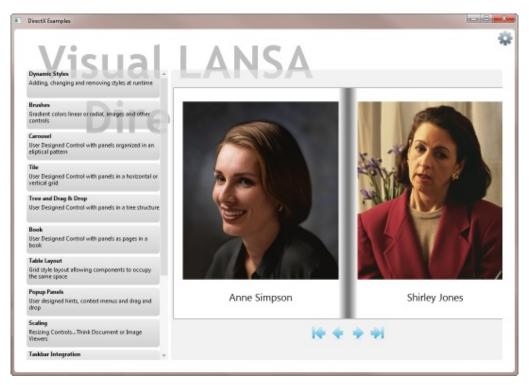


Click on the first image to activate the control buttons. Click on the *right* button to move through employees one at a time.



Note that only the *right* and *end* buttons are enabled, as you are currently at the beginning of the employee file.

2. Continue clicking on the right button to move through the employees *book*.



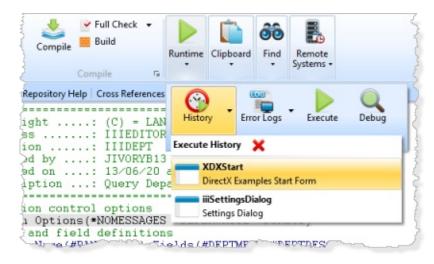
Note that the application displays a "dummy" picture when no employee image is held in the file PSLIMG.

3. Close the DirectX Examples form.

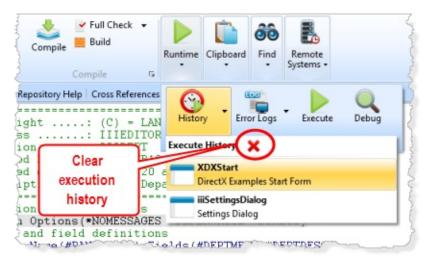
#### **Step 4. Execute the Application from the Execution History List**

In this step you will use the *Execution History* list to execute the XDXStart form again. This list shows forms and processes you have recently executed.

1. Click on the *History* button on the *Home* ribbon to open the *Execution History* list:



- 2. Select **XDXStart** from the drop-down list to select it to run again:
- 3. Close the form.
- 4. Note that *Execution History* can be cleared when required.



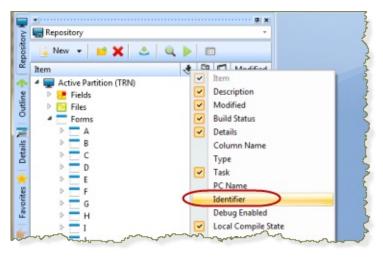
## Step 5. Execute the Form from the Windows Start Menu

In this step you will execute the application from the Windows start menu. When you run the form in this way, you are starting it as a normal Windows application, outside of the development environment.

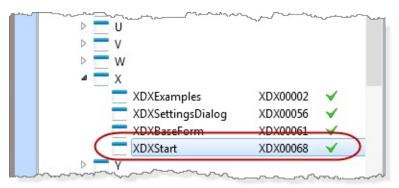
Icons are provided in the LANSA folder, which enable a form or a process to be run locally or in client server mode against the iSeries server.

When you connect in RDMLX mode, you can read and write data types such as String, Datetime and BLOB to and from the iSeries server.

1. Find the form XDXStart on the *Repository* tab. Right click on a column heading and ensure that the *Identifier* column is shown.



2. Note the *Indentifier* for the form XDXStart is XDX00068:



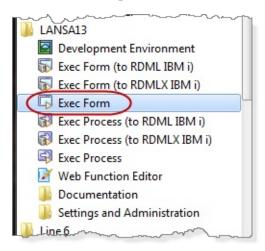
If Long Names are enabled at partition level, all Repository objects have both a *Name* and an *Indentifier*. If you opened XDXStart in the editor and selected the *Repository* tab, you would find the following information:

Definition	
▼ Name	XDXStart
Identifier	XDX00068
Description	DirectX Examples Start Form
Framework	Administration
Group(s)	
A Multilingual Details	
English	DirectX Examples Start Form
French	Formulaire de démarrage exemples DirectX

*Name* provides a long name which can be used to reference the object in your RDMLX code.

*Identifier* is assigned automatically when you create an object, must be unique within the partition and may be up to 9 characters long. The *Identifier* may be manually set, but only at create time.

3. From the Windows Start menu, expand LANSA and select Exec Form.



4. To execute the form XDXStart using the *Execute Form* dialog, you must use its *Identifier*, XDX00068. You must also execute it as a *DirectX* application (*Render Type* = X)

Form Name	XDX00068
Language	ENG
Partition Identifier	TRN**.
LANSA User	PCXUSER**
	X

Render Type		Execute Form on wor	kstation
<b>**</b> Use the values rele	evant to your situation.	Form Name Language	XDX00068 ENG
	5	Partition Identifier	TRN
		LANSA User	JIVORY13
		Database User	USERID
		Database Password	
		Database Name	LX2JOHNI
		Database Type	MSSQLS
		Default Printer	LPT1
		Debug	N
		Debug Host	johnivory-pc:51237
		Trace	N
		Max Trace Lines	20000
		Trace level	4
		Trace Categories	ALL
		Heap Validation	N
		Render Type	X
		Graphics Processing	Н
		OK Cancel	Help Parameter Help

- 5. If you have questions about any of these parameters, click *Parameter Help* button.
- 6. Press *OK* to execute the form.
- 7. Execute part of the DirectX Examples application and then close it.

### Summary

### **Important Observations**

- Both forms and processes are executed using the *Execute*... context menu option.
- Applications can be executed from within the Visual LANSA editor or from the desktop.
- A live application would start from a shortcut on the desktop

# **Tips & Techniques**

- You can also execute applications by pressing **Ctrl** + **Shift** + **E** or by using the toolbar icons.
- When run locally on your desktop, forms and processes are Windows DLL components which are loaded by the LANSA x_run.exe. X_RUN.EXE is part of the LANSA run-time system.
- Visual LANSA application can execute in client/server mode to access data on the IBM i.
- Icons in the LANSA folder enable applications to be started in client server mode against an iSeries server. A change to these settings would also enable an application to run against a Windows server.

# What I Should Know

- How to execute a LANSA application from within LANSA.
- How to execute a LANSA application from the Execution History list.
- How to execute a LANSA application from the desktop.

# **Repository Development Tutorials**

## What Are the Repository Development Tutorials?

The exercises in this tutorial have been written for new LANSA developers, so no experience with the LANSA Repository or RDML programming language is required. They are designed to introduce the fundamental repository skills required to build fields and files that will create your application database. Read REP000 - What is the LANSA Repository?

In order to do the exercises, you should be familiar with the Microsoft Windows user interface and it is recommended that you complete the Visual LANSA User Interface Tutorial before you start these exercises.

The following exercises are included:

**REP001 - Create Fields** 

REP002 - Field Visualizations

- REP003 Validation Rules
- REP004 System and Multilingual Variables

**REP005 - Creating Files** 

**REP006 - Logical Views** 

- REP007 File Validation Rules/Triggers
- REP008 Virtual Fields
- **REP009 Access Routes and Predetermined Join Fields**
- **REP011 Repository Summary**

REP012 - Check In Objects (Optional)

### **Before you Begin**

You must have LANSA Demonstration Personnel System installed in the partition that you will use with the set up options as described in What Partition Should I Use?.

The LANSA Demonstration Personnel System contains all the objects used by these exercises.

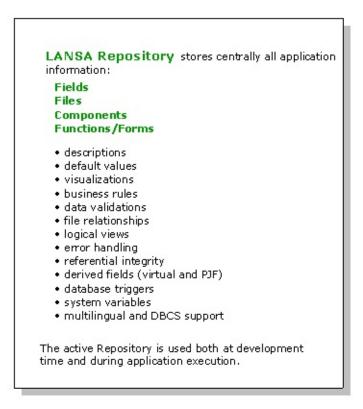
# **REP000** - What is the LANSA Repository?

# The Repository Concept

The LANSA *Repository* stores a vast range of information about applications in a central location.

LANSA's *Repository* architecture significantly reduces application coding because information is defined just once in a central location instead of being repeated wherever it is used in application programs. For example, a business rule for a field is defined once instead of in every program which uses that field.

In addition to bringing discipline to the development process, the repository also simplifies application maintenance because the centrally stored information is easy to find and change. Testing can be much faster and easier. Changes can often be made to the repository without impacting existing coded application logic.



The *Repository* stores information about fields (or elements) in an application such as descriptions, column headings, edit codes, visualizations, default values, help text, prompt programs, etc. It acts like a data dictionary for your application. It also stores components which can be shared by applications.

In addition, the *Repository* stores information about the files (or application database) such as physical files, logical views, file relationships, file definition attributes, file validation rules, trigger programs, etc.

Further, procedural information or business rules about the application in the form of validation rules, system variables, trigger and functions is stored in the repository.

Finally, LANSA offers some special *Repository* features such as multilingual definitions, virtual fields and predetermined join fields which simplify and accelerate development.

All information is stored in a non-platform specific format, which enables LANSA to build applications for iSeries, Windows and Linux deployment

### **REP001 - Create Fields**

The *Repository* field definitions are the foundation of your business applications. Other objects in the *Repository* such as files, components and functions are built from the basic field definitions. The centralization of a single definition and its reuse provide huge productivity benefits.

### **Objectives**

- To highlight how the LANSA *Repository* increases developer productivity.
- To create the following fields in the repository (where iii=your initials):

Field Name	Description	Type / Length
iiiDeptCode	Department Code	Alpha(3)
iiiDeptDescription	Department Description	Alpha(20)
iiiEmployNumber	Employee ID	Alpha(2)
iiiSurname	Employee Surname	Alpha(25)
iiiGivenName	Employee Given Name	Alpha(25)
iiiSalary	Employee Salary	Packed(11,2)
iiiStartDate	Employee Start Date	DateTime(26)
iiiEmployNotes	Employee Notes	String(512)

• These fields will be used to create the Department and Personnel files in later repository exercises.

To achieve these objectives you will complete the following:

Step 1. Prepare Your System

Step 2. Copy Fields

Step 3. Manually Create Fields

Step 4. Reference Fields

Step 5. Delete a field

Step 6. Create a Dynamic List for Your Fields

Step 7. Create and Execute Test Form

Step 8. Change your field definitions Summary

## **Before You Begin**

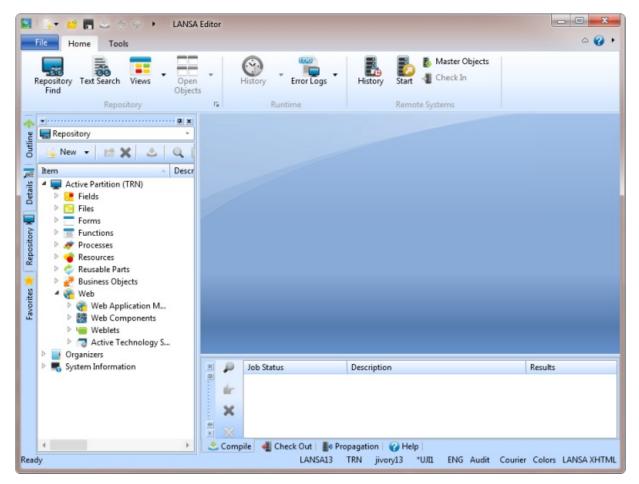
In order to complete this tutorial, the demonstration Personnel System files must have been installed in this partition. You should also have completed the *Visual LANSA User Interface Tutorials*.

### **Step 1. Prepare Your System**

- 1. Start the *Visual LANSA Development Environment* using the icon in your Program folder or on your desktop. (For details, refer to the *LANSA User Interface Tutorials*.)
- 2. Logon to the partition with a valid userid, password, partition, language and Task ID.

If you are using a Standalone Visual LANSA installation, you can log in with PCXUSER / PCXUSER.

- 3. If a warning message appears, select *Yes*. The message is simply a warning that you have the authority to change all the objects in the repository.
- 4. The Visual LANSA Editor will open. This is what it may look like:



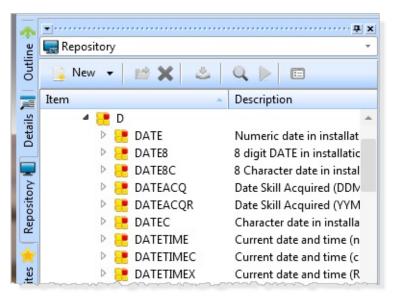
You can configure the Editor for your own preferences. There are many different settings and options. For more details, refer to the *Visual LANSA User Interface* (which you should have completed before starting these

tutorials).

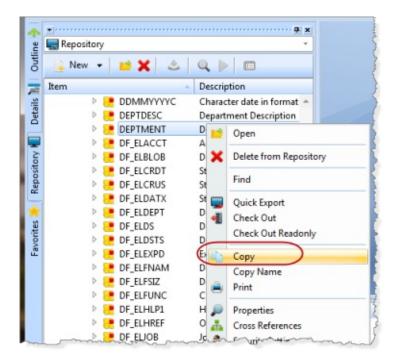
# **Step 2. Copy Fields**

In this step you will create fields called iiiDeptCode, iiiDeptDescription and iiiEmployNumber by copying the DEPTMENT, DEPTDESC and EMPNO fields that are already defined in the repository.

- 1. On the *Repository tab*, locate the field to be copied:
  - a. Expand the Fields node (if Alphabetic Groupings is on, open the list of fields for the letter **D**).



- b. Locate the **DEPTMENT** field. It is the code field used to identify departments.
- c. Right click on the DEPTMENT field to display the context menu. Select the *Copy* option.



A Create as copy of DEPTMENT dialog will appear.

Create as copy of D	EPTDESC	x
Name	iiiDeptCode	Create
Description	Department Code	Cancel
Initial Public Access	Normal *	
Identifier		Open in editor
Copy rules and trigger	s 🔽	
Copy visualization		
Copy help text		

- 2. Enter the basic details for the field in this dialog:
  - a. Enter a *Field name* of **iiiDeptCode** (where **iii** are your initials).
  - b. Leave the *Description* of **Department Code**.
  - c. Select (ü) the option to Copy rules and triggers.
  - d. Select (ü) the option to Open in editor.
  - e. Press the *Create* button.

#### Do not enter an Identifier. This will be generated.

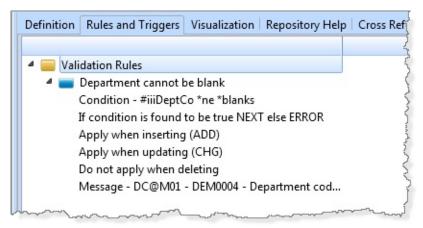
You can press F1 to access the online help for details about any of the field parameters.

▼ A red triangle in an input field indicates an error. Usually this happens when the value in the field is blank or incomplete. To see the error message click on the red triangle.

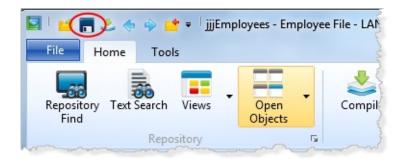
- 3. The **iiiDeptCode** field is opened in the LANSA Editor to allow you to edit other field characteristics.
  - a. Double-click Length. In the Details tab, change Field Length to 3.

-	*			······································	Definition	n Rules and Triggers	Visualization	Repository Help	Cross References	8
e l	Repository	/		*						
Outline	New -	1	📁 🗙 🗠 🕹		4 🔻 📒	Details		DeptCode		^
10	Item		A.	Description		entifier		DEPTCO		
Repository 🚺 Details			DAYC DDMMYY DDMMYYC DDMMYYYY DDMMYYYYC DEPTDESC	Current day (character) Numeric date in format I Character date in format I Numeric date in format I Character date in format Department Description	Ty Le Di	rpe ingth icimals efault Value iference Field	A 4	Iphanumeric BLANKS		
Repo			DEPTMENT DF ELACCT	Department Code Account		escription		epartment Code		
*	Þ	ē	DF_ELBLOB	Document		ibel eading 1		epartment Dept		
vorites	Þ		DF_ELCRDT DF_ELCRUS	Stamp - Create Date/Tim Stamp - Create User		eading 2	(	Code	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

b. Select the Rules and Triggers tab. Notice that there is one validation rule copied from the DEPTMENT field.



c. Press the *Save* button on the Editor toolbar to save the field.



d. Close the field.

Visualization   Repository Help	Cross References	
e blank		
o *ne *blanks to be true NEXT else ERROR		
(ADD)		

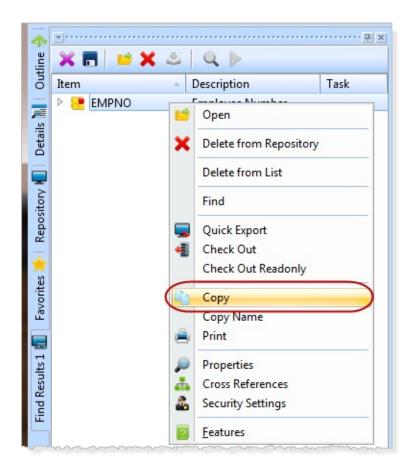
- 4. Create the **iiiEmployNumber** field by copying the EMPNO field.
  - a. Locate the EMPNO field using the *Find* dialog by pressing the *Repository Find* button on the toolbar.



b. Check that *Field* is selected in the Object Type list. In the *Find Text* field enter **empno** and press the *Find* button.

bject Types	Find Text	Find
ActiveX Bitmap Business Object Cursor	empno In name In description	Cancel
External Resource	Filters	
Field	Task ID:	
File	User:	
Form		
Function	Referring to:	
Icon	Status:	
Multilingual Variable	Framework:	*
Primitive	Group:	*
Process	Modified after:	*
Reusable Part System Variable	Modified before:	*
Template		
Visual Style		
Web Application Mod Weblet		
Web Component	Results	
Web Service Proxy		
.NET Component	Show results in: New Tab Sheet	· · ·

- c Close the *Repository Find* dialog.
- d. In the *Find Results 1* tab right click on the **EMPNO** field to display the context menu. Select the *Copy* option.



- e. The Create as copy of EMPNO dialog will appear.
- f. Enter a *Field name* of **iiiEmployNumber** (where **iii** are your initials).
- g. Do not copy the rules and triggers, visualization or help text.
- h. Do not select (ü) the option to Open in editor.
- i. Press the *Create* button.

The field is created in the Repository but is not opened in the Editor.

5. Create the **iiiDeptDescription** field by copying the DEPTDESC field.

a.Locate the field DEPTDESC on the *Repository* tab.

- b. Copy the rules and triggers.
- c. Select *Open in editor* so that you can edit it once it has been created.
- 6. Specify that the department description can be entered in lower case.
  - a. Double-click on the Input attributes heading in the field definition to open the *Details* tab.

b.Select LC – Lowercase Entry Allowed

- c. Save and close the field definition.
- 7. On the *Favorites* tab, select the Last Opened tab. Notice that the iiiDeptCode and iiiDeptDescription fields are listed but the iiiEmployNumber field is not. When you open any object in the Editor, it will be added to the Last Opened list.

### **Step 3. Manually Create Fields**

Field Types: Character fields in LANSA are usually Alpha (up to 256 characters) or if longer, String fields. Numeric fields are most commonly defined as Packed. It is recommended that dates and times are defined as DateTime fields.

In this step you will manually create three fields called iiiSalary,iiiStartDate and iiiEmployNotes.

- 1. The first field you will create is the **iiiSalary** packed field.
  - a. Use the *New* button on the *File* menu and select *Field*.

The *New field* dialog will appear.

b. Enter the following characteristics for the field:

Field name	iiiSalary
Description	Salary
Field type	Packed
Field length	11
Decimals	2

😬 New Field	-		×
Name	iiiSalary		Create
Description	Salary		Cancel
Field Type	Packed	•	
Field Length	11 ‡		Open in editor
Decimals	2 \$		Close
Reference Field			
Identifier	<b>III</b> SALARY	]	
Enabled For RDMLX			

c. Select (ü) the option to *Open in editor*.

- d. Select (ü) the *Close* option to close the *New Field* dialog. (When you are creating many fields in one go, it is convenient to keep the dialog open.)
- e. Press the *Create* button to open the field definition in the Editor.
- 2. Now that the field is open, you can edit all its characteristics.
  - a. In the Definition tab double-click Default value. In the *Details* tab enter a default value of 10000.

Details		Definition Rules and Trigg	gers   Visualization   Repository Help
Name	jjjSalary	🖉 🦉 Details	
Identifier	JJJSALARY	Name	jjjSalary
Field Type	Packed	- Identifier	JJJSALARY
Field Length	11	÷ Type	Packed
Decimals	2	÷ Length	11
Default Value	10000	Decimals Default Value	2
Reference Field		Reference Field	
Descriptions		Description	Employee Salary
Description	FSalane		

b. Click on Edit Mask. Set its value to 2.

Reposi	Descriptions			Reference rield		20
Rep	Description	Salary		Description		Salary
-	Field Label	Salary		Label		Salary
~		Salary		Heading 1		Salary
Lite	Column Headings	Salary		Heading 2		3
Favorites				Heading 3		2
-				Edit Mask		1
155	Details			Keyboard Shift		ļ
ts 1	Details			Enabled For RDMLX		No
esu	Edit Mask			System Field		No
Find Results	Keyboard Shift	A		Prompt Process		i
E	Enabled For RDMLX	1 1,234,567,12300 2 1,234,567,12300 (Blank when zero)		Prompt Function		
	LINDIEU FOI KDIVILA	2 1,254,307.12300 (Blank When Zero) 3 1234567.12300		Alias Name		}
	System Field	4 1234567.12300 (Blank when zero)				2
	Prompt Function	A 1,234,567.12300CR	4	🥪 Input Attributes		Ę.
	D 10	B 1,234,567.12300CR (Blank when z		FE		Field exit key
	Prompt Process	C 1234567.12300CR		~~_RB	Jan Martin	Richt-rliggt

- c. The *Edit Mask* controls how the field is formatted when displayed. The drop-down for the *Edit Mask* shows samples of the edit masks. You can also press F1 to review the help text.
- d. Leave all other field characteristics as their defaults.
- e. Save the iiiSalary field definition.

- f. Close the field definition.
- 3. Next, you will create a DateTime field which will be used to store the date an employee joined the department.

Select the *New* option on the *File* menu and select *Field* from the menu options.



The *New field* dialog will be displayed.

4. Enter the following characteristics for the field:

Field name	iiiStartDate
Description	Start Date
Field type	DateTime
Field length	26
Enabled for RDMLX	Yes

- a. Deselect the option to Open in editor.
- b. Uncheck the *Close* option so that the *New field* dialog will appear again.
- c. Press the *Create* button.

The field has been created in the Repository but has not been opened in the Editor.

- 5. Next you will create a string field to store employee notes.
  - a. The *New field* dialog will appear again.
  - b. Enter the following characteristics for the field:

Field name	iiiEmployNotes
Description	Notes
Field type	String
Field length	512
Enabled for RDMLX	Yes
Close	Yes

c. Press the *Create* button.

You have now manually created fields Salary, Start Date and Notes.

#### **Step 4. Reference Fields**

Reference Fields: A field inherits these characteristics from its reference field:

- Type
- Length and decimal positions
- Edit mask and word
- Input and output attributes
- RDMLX enabled flag
- Default value
- Keyboard shift

When the definition of a reference field is changed, the characteristics of all fields that are based on it, are changed immediately.

In this step you will create two fields called iiiSurname and iiiGivenName. These fields will use the reference field feature of the LANSA Repository so that their characteristics are inherited from the shipped STD_NAME field.

1.On the *Repository* tab, find the field STD_NAME and *Copy* it to create a new field iii_Name.

a.Copy the Rules and Triggers

b.Do not open the field in the editor.

2. Create the iiiSurname field using the New button on the *File* menu. Choose Field from the graphical menu.

😬 New Field			-	×
Name	iiiSurname			Create
Description	Surname			Cancel
Field Type	Alpha		<b>•</b>	
Field Length	25	\$		Open in editor
Decimals		¢		Close
Reference Field	iii_NAME		Standard NAME	
Identifier	IIISURNAM			
Enabled For RDMLX				

a. In the New field dialog, enter:

Field name	iiiSurname
Description	Surname
Reference Field	III_NAME
Close	Yes
Open in Editor	Do not select.

b. Press the *Create* button.

Notice that the field type and length are derived from the reference field.

- 3. Select the *New* button from the File menu and choose *Field* from the grahical menu to create the Given Name field (iiiGivenName). Alternatively you could use the Alt + F + N keys.
  - a. In the *New Field* dialog enter the following characteristics:

Field name	iiiGivenName
Description	Given Name
Field type	Packed
Field length	15

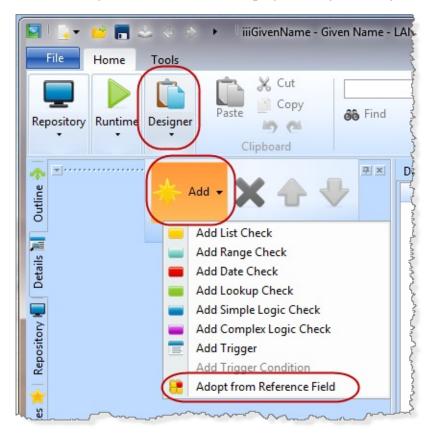
- b. Do not specify a *Reference Field* yet.
- c. Select the option to *Open in editor*.
- d. Select the option to *Close* so that the dialog will be closed.
- e. Press the *Create* button.
- 4. The **iiiGivenName** field is opened in the Editor.
  - a. Click on Reference Field in the *Definition* tab to open the *Details* tab.
  - b. In the *Details* tab type in iii_NAME as the *Reference Field*.

Details		Definition Rules and Triggers   Visualization   Repository Help   Cross References
Name	iiiGivenName	4 📜 Details
Identifier	IIIGIVENN	Name iiiGivenName
Field Type	Alpha	Identifier IIIGIVENN
Field Length	25 \$	Type Alphanumeric
Decimals	÷	Length 25
Default Value	*BLANKS	Decimals Default Value *BLANKS
Reference Field	III_NAME ··· Standard NAME	Reference Field II_NAME
Descriptions		Give bloge when the state

The iiiGivenName field is now based on the definition of the III_NAME field. Notice, for example, that its field type is now Alpha and its length is set to 25. These characteristics are inherited from the reference field and cannot be changed.

c. Review the Rules and Triggers tab.

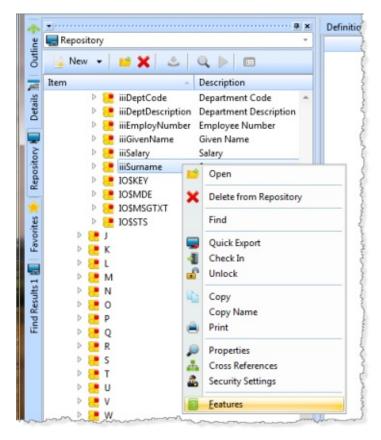
Notice that there are no rules. However, if you click on the *Add* button on the *Home* ribbon, you will see the *Adopt from reference field* option.



#### Do not adopt any rules at this stage.

- d. Save and close the field.
- 5. Open the III_NAME field.

- 6. Change the Field length to 20.
- 7. Save the III_NAME field definition.
- 8. Locate the iiiSurname field on the *Repository* tab. Right-click on the field and choose the *Features* option from the context menu.



The *Details* tab will be displayed, showing the field's definition:

Features	
(a + a) +   ≤ ≤   Q ≥	E • 9
Classes: iiiSurname	. •
▲ Field	
📜 iiiSurname	Surname
4 Definition	
Туре	Alphanumeric(20)
<ul> <li>Reference field</li> </ul>	
📒 III_NAME	Standard NAME
Default Value	*BLANKS
Input Attributes	
LC	Lowercase entry allowed
Label	Surname
Column heading	Surname
Enabled for RDMLX	No
Class	
4 📒 iiiSurname	Surname
www.altertowner.	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

- 9. Change the Field length of III_NAME back to 25.
- 10. Save and close the III_NAME field.

### Step 5. Delete a field

- 1. In the *Repository tab*, locate the III_NAME field.
- 2. Right click on the III_NAME and select *Delete from Repository* from the context menu.

**Note:** You will not be actually deleting this field.

You will be asked to confirm the deletion.

Confirm delete ob	ject	×		
Name	Description Standard NAME	Delete Cancel		
Delete from host repository				

3. Press *Delete* so that you will see an Integrity Check Failure dialog will appear to warn you that this field is required. (The field has not been deleted yet.)



There are two reasons why the Integrity Check Failure occurs: firstly because III_NAME is a system field and secondly because it is used as a reference field.

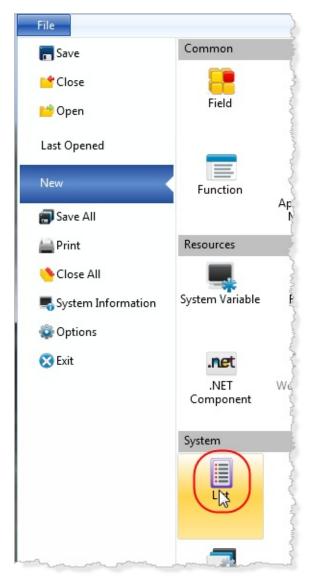
- **Note:** System fields cannot be deleted from the repository unless they are specifically changed to non-system fields. For further details about system fields refer to the F1 Help from the Field's *Details* tab.
- 4. Press *No*. Do NOT delete the field.

### Step 6. Create a Dynamic List for Your Fields

Dynamic Lists: A dynamic list will include any items that satisfy the inclusion criteria.

In this step you create a dynamic list for the fields and other objects you create in this tutorial. The list will include all objects starting with your initials, so as you create objects they are automatically added to the list.

1. From the *File* menu, select the *New* option and choose *List* from the graphical menu.



2. In the *New List* dialog specify:

- a. Enter iiiLIST as the name (where iii corresponds to your initials).
- b. Select **Dynamic** as the list Type.
- c. Store the list as a **User list**.
- d. Select the Is Favorite option to add the list to the Favorites tab.
- f. Press *Create* to continue.

New List		<b>x</b>
Name	iiiList	Create
Туре	Dynamic 🔹	Cancel
Stored As	User List 🔹	✓ Is Favorite
Identifier	OL0	- Bratonic

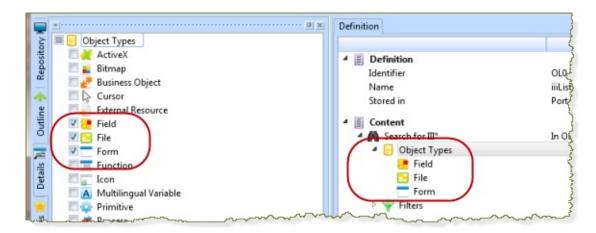
3. On the *Definition* tab, double click on *Search for* to expand it. The *Find Text* dialog will open on the left tab. Enter **III*** in *Find Text*.

Find text will convert text to upper case and the search ignores case.

Details		Definition
Find Text In Name		Definition     Identifier
In Description		Name Stored in
		<ul> <li>✓ M Search for Ⅲ*</li> <li>▷ ⊖ Object Types</li> <li>▷ ❤ Filters</li> </ul>

4.Click on *Object Types* and select **Field, File** and **Form** in the *Object Types* list shown on the left hand panel.

Notice that your selection is reflected in the dynamic list *Definition* in the right panel.



- 5. Save and close the list.
- 6. Notice that the iiiLIST is now included on the *Favorites* tab.

Notice that all the fields starting with your initials are included in the list. As you have created no forms or files yet, none have been selected.

Favorites			
🖻 🗙 🕹 🔍 🕽			
🔚 Weblet Templates	🗐 La	ast Op	pene 🛛 📳 iiiList
Item 🔺	8	Ø	Description
📑 iiiBasicWinContr		2	Basic windows Contr 🔺
IIIBVA10		2	Manage Vehicle Recc
iiiDepartments		1	Departments Table
👂 🛅 iiiEmpByDeptView			?
iiiEmpHolByCode			Employee Holidays B
🕨 📴 iiiEmpHolidays		1	Employee Holidays
🕑 📴 iiiEmpImages		9	Employee Images

### Step 7. Create and Execute Test Form

In this step you create a simple test form which will show the fields you have created on a form. No coding is required to create this application.

1. From the *File* menu, select the *New* option and select *Form* from the graphical menu. Then select *Basic Form*, to display the *New form* dialog.



2. Enter the following characteristics for the form:

Name	iiiiTestFields
Description	Test Fields
Framework	Select any suitable Framework such as
	Personnel & Payroll.
Enabled for RDMLX	<b>Yes</b> (the default for this option is determined by the partition settings).

New Form			×
Name	iiiTestFields		Create
Description	Test Fields		Cancel
Framework	Training Development (TRAINING)		Cancer
Group			
Identifier	ШТЕSTF		
Enabled For RDMLX			

2. Press the *Create* button. A blank form will open in the Editor.

S
::
• •
• •
• •
• •
• •
 • •

a. Drag all the fields. Except III_NAME from iiiLIST to the form. You can either drag them individually or as a group.

If you select them as a group using the Shift + Left Mouse button, the fields will be lined up neatly on the form.

b. Your form might appear like the following:

		x
Department Code	ABC	
Department Description	aAbBcCdDeEfFgGhHiljJ	
Employee Notes Employee ID	aAbBcCdDeEfFgGhHiljJkKILmMnNoOpPqQrRsStTuUvVwWxX AB	(y YZZak
Given Name	aAbBcCdDeEfFgGhHiljJkKlLm	
Salary	123,456,789.12	:::::
Start Date	01/01/1900 00:00:00	* : : :
Surname	aAbBcCdDeEfFgGhHiljJkKILm	::::

Do not add any code to your form. You are simply testing the interface.



- 4. Press the **Compile** button on the *Home* ribbon to compile your form. The Compile tab will show you the progress of the compile.
- 5. Check that the form compiled successfully.

×	P	Job Status	Description	Results	Currently Proc	essing	Started	Ended	
<u>문</u>		Completed	iiiTestFields - Test Fields	Compiled 1 of 1			05/12/2012	. 05/12/	2012
1	r.								
-									
	2								
	×								
*		nt 👶 Compile 🐗	Check Out   📑 Propagation   🛼 Te	ext Search   🕜 Help					

6. Press the *Execute* button from the *Runtime* group on the *Home* ribbon, to execute the form.



**Note:** The appearance of ribbons depends on the size of the Visual LANSA form. Groups such as *Runtime* are displayed as a single button when space is limited.

The execute *Form as Windows Application* dialog may appear depending on your option settings.

Form as Windows Ap	plication
Default Printer	LPT1
Debug	N
Debug Host	johnivory-pc:51237
Trace	N
Max Trace Lines	20000
Trace level	4
Trace Categories	ALL
Heap Validation	N
Show Command Lin	e N
Render Type	X
Graphics Processing	Н
OK Cancel	Help Parameter Help

7. Press the *OK* button to execute the form.

Your window should appear something like following:

Designed		
Department Code		
Department Description		
Employee Notes		
Employee ID		
Given Name		
Salary		
Start Date		
Sumame		

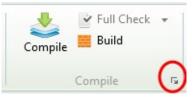
- 8. Test the following:
  - a. Enter the number 2000 in the Salary field. Notice how the number is formatted once focus leaves this field (this is controlled by the *Edit Mask* in the field definition).
  - b. Drop down the calendar in the *Start Date* field to specify a date. The way the *Start Date* field is displayed is the default visualization for a *DateTime* field (this will be described in more detail in REP002 Field Visualizations).
- 9. Close the Test Fields form.

### **Step 8. Change your field definitions**

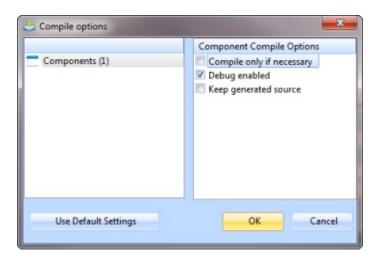
In this step you will make minor changes to the field definitions based on the results of your test.

Note: An application must always be recompiled after you have changed field attributes that impact their display within the application.

- 1. Using the iiiLIST tab, open the iiiSalary field.
  - a. Set the Default value to *ZERO.
  - b. Change the Description to 'Monthly Salary'.
  - c Save and close the field.
- 2. Using the iiiLIST tab, open the iiiEmployNumber field.
  - a. Set the Field Length to 2.
  - b. Save and close the field.
- 3. Recompile your Field Test application (**iiiTestFields** form).
  - a. Switch to the open iiiTestFields form in the editor.
  - b. On the *Home* ribbon in the compile group click on the button shown to open the *Compile Options* dialog,



c. Deselect the *Compile only if necessary* option.



- d. Press the *OK* button to submit the compile.
- 4. Execute your Field Test application and notice the change you have made to the *Monthly Salary* label has been actioned.

Department Code		
Department Description		
Employee Notes		
Employee ID		
Given Name		
Monthly Salary		
Start Date		
Sumame		

- a. Enter an employee number. Notice it can only be two characters long.
- b. Notice the Monthly Salary field is blank when form is first executed.
- 5. Exit the form.
- 6. Close all objects in the Editor.

Note: You can *Close All* from the *File* menu.



#### Summary

#### **Important Observations**

- There are many different ways to create a field.
- Basic field definitions can be quickly created directly from the *New field* dialog. Fields can be opened in the Editor to specify detailed field characteristics.
- If a field is created but not opened in the Editor, it will not appear in your Last Opened list.
- The *Create as copy of field* dialog allows a one-time copy operation to create a field.
- Reference fields create a permanent link between two fields. Changes to field types, lengths, etc. are inherited. Not all field characteristics, such as descriptions and labels, are inherited.
- The LANSA Repository performs referential integrity checks whenever you attempt to delete an object. For example, you will be informed if a field is used in a file or function before you proceed with deletion.
- For your test application notice that:
  - The form/dialog was designed using the information from the repository.
  - Default values are displayed based on the information from the repository.
  - Input and output characteristics are based on the information from the repository.
  - When changes were made to your field definitions, the application was recompiled to update it with the definitions.

## **Tips and Techniques**

• Use reference fields when you need a permanent relationship between fields instead of a one time copy.

## What I Should Know

- How to define a field by copying an existing repository field.
- How to manually define a field.
- How to change attributes that affect the display on dialogs and forms.
- How default values can be used by your applications.

### **REP002** - Field Visualizations

You can choose how a field is shown in an application. For example a numeric field can be visualized as a simple entry field, as an entry field with spin buttons, as a progress bar or a track bar.

### **Objectives**

- To introduce the basics of field visualization. (You will do more advanced examples of coding Field Visualizations when you do the FRM tutorial.)
- To show how field visualizations can be defined.

To achieve these objectives you will complete the following:

Step 1. Review iiiStartDate Field Visualization

Step 2. Review iiiEmployNotes Field Visualization

Step 3. Review iiiSalary Field Visualization

Step 4. View the Test Form

Step 5. Execute the Form

Step 6. Adjust the Form

Summary

## **Before you Begin**

In order to complete this exercise, you should have completed the previous exercise.

## Step 1. Review iiiStartDate Field Visualization

In this step you will review the standard field visualization created for DateTime field type and you will add another visualization to the iiiStartDate field.

A field can have several visualizations defined for it. The default visualization which is used is determined by its DefaultVisualization property.

- 1. Open the iiiStartDate field in the Editor.
- 2. Select the *Visualization* tab.

You will notice that there is a visualization VisualDateTime already defined which shows a Datetime Entry Field. This is the way the Start Date field was displayed (edit box with a drop-down calendar) in the Test Fields application you created in the previous exercise.

(If your partition is web-enabled, the field may also have a default weblet visualization.)

3. Press the *Calendar* 😰 toolbar button to add a calendar visualization to the **Start Date** field.

× # * * * *	🖻 🧿 🖱 🗮 🍮 🔹 🖷 🖗	↑ ↓
Field Parts		
Visualizations	Start Date 0	01/01/1900 00:00:00
VisualDateTime		
🖰 VisualCalendar		44 4 January 1900 + ++
		Mon Tue Wed Thu Fri Sat Sun
		25 26 27 28 29 30 31
	· · ·	1 2 3 4 5 6 7
	Start Date	8 9 10 11 12 13 14
	and a second	15 16 17 18 19 20 21
		22 23 24 25 26 27 28
	111	<b>29 30 31</b> 1 2 3 4
	100	Today: 05/12/2012

Notice that the field now has two visualizations: VisualDateTime and VisualCalendar.

- 4. Double-click the **VisualCalendar** in the *Field Parts* list to display its properties in the *Details* tab.
- 5. Locate the DefaultVisual property and notice it is set to False.

💂 Repository   🐴 Ou	tline 📜 Details 🔶 Favo	orites   👉 💶
		·····
VisualCalendar		-
<b>•</b> ?		
Properties Events M	ethods	
Caption		*
Cursor	*NULL	
C DefaultVisual	False	
P DragStyle	None	
Ellinses	None	

- 6. Double-click the VisualDateTime visualization to display its properties. Its DefaultVisual property is **True** which means this is the default way the field will be displayed on a form.
- 7. Do not change the DefaultVisual property. You want to keep the Date Entry Field drop-down visualization because it takes up less room than the calendar on the form.
- 8. Notice that the VisualDateTime shows both the date and time. You will not need to see the time in the Start Date field. To hide it set the *ShowTime* property to **False**.

sualDateTime		
roperties Events Method	ds	_
PrompterTabStop	True	4
ReadOnly	False	
Rotation	0	
RotationOriginLeft	50	
RotationOriginTop	50	
ScaleHeight	100	
ScaleOriginLeft	50	
ScaleOriginTop	50	
ScaleWidth	100	
ShowDate	True	
ShowDateButton	True	
ShowDateButtonImage	*NULL	
ShowError	False	
ShowPrompter	False	
ShowSelection	False	-
ShowTime	False	
ShowTimeButton	False	
SizingScheme	*NULL	
Skewhat	h	

Although LANSA also has a field type Date, the use of the DateTime type is recommended because LANSA manages DateTime fields if you move the data from one time zone to another. Typically, data can be represented in the local time zone but saved in the database as UTC (Universal Time). 9. Save and close the field.

### Step 2. Review iiiEmployNotes Field Visualization

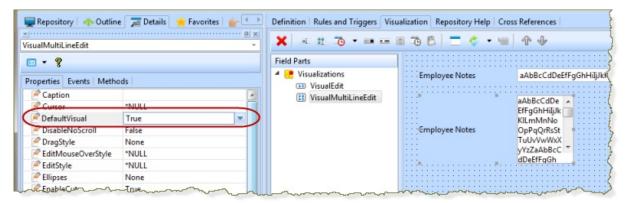
In this step you will review the field visualizations created for the iiiEmployNotes field. You will add a multi-line edit box and make this the default visualization.

Multi-line edit box is often the best visualization to use for long character fields.

- 1. Open the **iiiEmployNotes** field in the Editor.
- 2. Select the *Visualization* tab.

You will notice that the standard visualization for a String field is simply an entry field.

- 3. Press the *Multi-line Entry Field* 🔢 toolbar button to add a multiple line edit box visualization for the Notes field.
- 4. Double-click the VisualMultilineEdit visualization to display the *Details* tab. Change its *DefaultVisual* property to **True.**



By changing the *DefaultVisual* property to **True** for the multiline edit box, the *DefaultVisual* property of *VisualEdit* will be set to **False** because only one visualization can be the default.

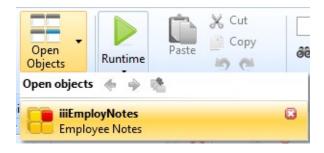
5. Make the *Width* property of the VisualMultiLineEdit to **400**.

Visible	True
🖻 VisualStyle	*NULL
🖻 VisualStyleEdit	*NULL
VisualStyleOfParent	True
Width	400

The default visualization for the Notes field is now a multi-line edit box.

6. Save and close the field.

Note: You can close any open object from the *Open Objects* view.



### Step 3. Review iiiSalary Field Visualization

In this step you will review the field visualization created for the iiiSalary field and you will add a spin edit box and make this the default visualization for the field.

Numeric fields can be visualized as simple entry fields, spin edit boxes, progress bars or track bars.

- 1. Open the **iiiSalary** field in the Editor.
- 2. Select the Visualization tab.

You will notice that the standard visualization for a numeric field is simply an entry field.

3. Press the *SpinEdit* toolbar button to add a spin edit visualization. The spin edit box allows the end-user to adjust the numbers using the spin buttons or by using the Up and Down keys.

🗙 🗉 🖞 🔞 • 🚥 🖬	ීම 🖺 🚍 🗇 🖷	<b>↑ ↓</b>	
Field Parts			
<ul> <li>Visualizations</li> <li>VisualEdit</li> <li>VisualSpinEdit</li> </ul>	Monthly Salary	123,456,789.12	
			:::
	Monthly Salary	123,456,789.12	

4. Change the DefaultVisual property of the VisualSpinEdit to True:

🗉 • 🤋		
Properties Events Meth	hods	
AutoSelect	True	
AutoTab	False	1
Caption		
Cursor	*NULL	_
PefaultVisual	True	-

5. Save and close the field.

### Step 4. View the Test Form

In this step you will review the test form and make sure the iiiNotes field fits on the form.

- 1. Open the iiiTestFields form.
- 2. Display the *Design* tab.
- 3. Resize and rearrange the form so that the Notes field fits properly.

E	Department Code	ABC
	Department Description	aAbBcCdDeEfFgGhHiljJ
E	Employee ID	AB
	Employee Notes	aAbBcCdDeEfFgGhHiljJkKILmMnNoOpPqQ rRsStTuUvVwWxXyYzZaAbBcCdDeEfFgGhH iljJkKILmMnNoOpPqQrRsStTuUvVwWxXyYz ZaAbBcCdDeEfFgGhHiljJkKILmMnNoOpPq QrRsStTuUvVwWxXyYzZaAbBcCdDeEfFgGh HiljJkKILmMnNoOpPqQrRsStTuUvVwWxXy YzZaAbBcCdDeEfFqGhHiljJkKILmMnNoOp
	Monthly Salary	123,456,789.12
	Given Name	aAbBcCdDeEfFgGhHiljJkKILm
	Start Date	01/01/1900
	:Sumame	aAbBcCdDeEfFgGhHiljJkKlLm

#### Note:

- a. A selected field can be moved using the Ctrl + Cursor keys.
- b. Fields can be moved as a group by selecting them with the Shft + Left Mouse or Ctrl + Left Mouse button.
- 4. Compile the form.

### **Step 5. Execute the Form**

In this step you will test your field definitions by executing the iiiTestFields form.

1. Press the *Execute* toolbar button in the Editor to execute form iiiTestFields.

The execute *Form as Windows Application* dialog may appear depending on your options.

This dialog will appear if the *Prompt for additional information* option was selected when executing from the *Verify* menu. The last options used in *Verify* / *Execute* are remembered, until you change them.

Form as Windows Application			
Default Printer	LPT1		
Debug	N		
Debug Host	johnivory-pc:51237		
Trace	N		
Max Trace Lines	20000		
Trace level	4		
Trace Categories	ALL		
Heap Validation	N		
Show Command Line N			
Render Type X			
Graphics Processing	Н		
OK Cancel	Help Parameter Help		

Press the *OK* button to execute the form.

2. Your form should appear something like this:

Test Fields	
Department Code Department Description	
Employee ID	
Employee Notes	
Monthly Salary	÷
Given Name	
Start Date	·
Surname	

- 3. Test your new fields:
  - a. Enter a large amount of text into the Notes multi-line entry field. Notice the scroll bar appears.
  - b. Adjust the salary using the spin buttons or the Up and Down buttons. Notice how the value is formatted when you move the focus away from the field.
  - c. Change the Start Date using the calendar.
- 4. Close the form.

### Step 6. Adjust the Form

In this step you will make some adjustments to the iiiTestFields form to place the fields in a logical order.

1. By dragging and dropping your entries, adjust the layout of the form to look something like this:

Department Code	ABC	
Department Description	aAbBcCdDeEfFgGhHiljJ	
Employee ID	AB	
Surname	aAbBcCdDeEfFgGhHiljJkKILm	
Given Name	aAbBcCdDeEfFgGhHiljJkKILm	
Monthly Salary	123,456,789.12	
Start Date	01/01/1900	
Employee Notes	aAbBcCdDeEfFgGhHiljJkKlLmMnNoOpPqQ RSStTuUV/wWxXyYzZaAbBcCdDeEfFgGhH ilJlkKlLmMnNoOpPqQrRsStTuUV/wWxXyYz ZaAbBcCdDeEfFgGhHiljJkKlLmMnNoOpPq QrRsStTuU/VwWxXyYzZaAbBcCdDeEfFgGh HiljJkKlLmMnNoOpPqQrRsStTuU/VwWxXy YzZaAbBcCdDeEfFgGhHilJJkKlLmMnNoOp	

Note that you can select several fields at a time and drag them together by selecting fields while holding down the Shift key.

Fields can also be moved using the Ctrl + Cursor keys.

2. Compile and execute your form.

Test Fields	- 0 ×
Department Code	
Department Description	
Employee ID	
Surname	
Given Name	
Monthly Salary	\$
Start Date	*
Employee Notes	

3. Close the *Test Fields* application.

#### Summary

#### **Important Observations**

- Field visualizations can be used in LANSA components such as forms, reusable parts, WAMs, etc.
- Fields types such as Alpha, String, Packed and Signed have a simple entry field as the default visualization, but many other types of visualizations, such as Radio buttons, drop downs, spin buttons, and other visualizations can be added.
- Field types such as DateTime have specialized visualizations that include features like calendar prompts.
- You can control which visualization is used by default. In the example of the iiiEmployNotes field, you would want a multi-line entry field to be used instead of a single entry field. Hence, the multi-line field was defined as the default visualization.

## **Tips and Techniques**

- Field visualizations ensure a consistent presentation of your fields.
- Field visualizations can be extremely powerful in the case of simple list selections. For example, a field GENDER can be visualized as Male and Female radio buttons. By defining this visualization once, it is automatically available to all components.
- For detailed examples of coding Field Visualizations, refer to the *Visual LANSA Windows Application Tutorials*.

## What I Should Know

- How to create a field visualization.
- How to perform simple changes to the field visualization definitions.
- How to set the default visualization.

#### **REP003** - Validation Rules

Validation rules are enforced by business rules defined in the Repository. Rules which are defined in the field definition become global rules, meaning that they will be enforced in any file where the field is used. Most rules are defined in the file definition in which case they are enforced only in the context of that file. You will learn more about this feature in the exercise REP007 - File Validation Rules/Triggers.

#### **Objectives**

- To show field level validation rules in the repository.
- To show how rules are adopted from reference fields.
- To add the following business rules to the repository:

iiiSurname Cannot be blank (copied from reference field)

iiiSalary Must be greater than 0

To achieve these objectives you will complete the following: Step 1. Review Existing Rule for iiiDeptCode Field Step 2. Adopt Rule from Reference Field Step 3. Create a Rule for the iiiSalary Field Summary

## **Before you Begin**

In order to complete this tutorial, you must have completed the previous tutorials.

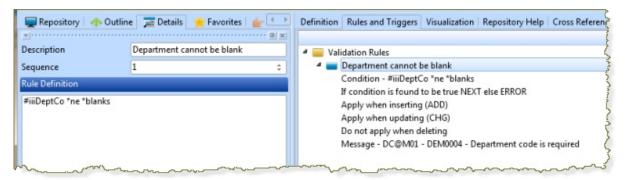
## Step 1. Review Existing Rule for iiiDeptCode Field

In this step you will review the validation rule defined on the iiiDeptCode field that you created in exercise REP001 - Create Fields. When you created the iiiDeptCode field, you copied the validation rules from the DEPTMENT field.

- 1. Open the **iiiDeptCode** field in the Editor.
- 2. Select the Rules and Triggers tab.

All validation rules defined on the field are displayed. **Note:** If the rule does not appear, then you did not copy it when you created the field. To correct this problem, delete the iiiDeptCode field and recreate it as described in Step 2. Copy Fields in REP001 - Create Fields.

3. Select and expand the existing rule Department cannot be blank by clicking the plus (+) to display the Details tab. This rule is a simple logic rule that ensures that the field is not blank.



There are different types of validation rule which each have their own set of parameters. A simple logic rule allows a simple expression to be evaluated when the value of a field has been received.

4. Review the Details to see the actions defined for this rule.

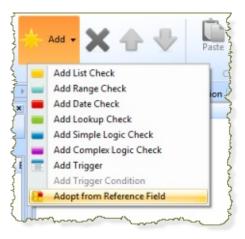


5. Do NOT close the field.

## Step 2. Adopt Rule from Reference Field

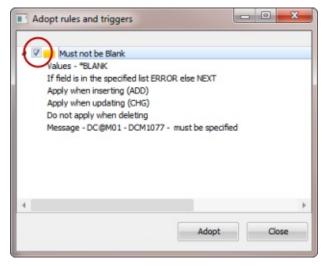
In this step you will add a rule to the iiiSurname field by adding the rule from its III_NAME reference field.

- 1. Open the **iiiSurname** field in the Editor.
- 2. Select the Rules and Triggers tab.
- 3. Click the Add button on the *Home* ribbon to open the *Add* menu and select the Adopt from Reference Field option.



The *Adopt rules and triggers* dialog displays the rules that iiiSurname can adopt from the III_NAME field. In this case there is only one.

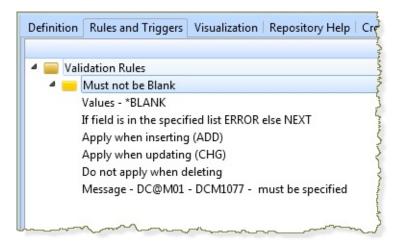
4. Select the *Must not be Blank* rule and press the *Adopt* button.



**Note:** If you press the Adopt button again, the rule will be adopted again.

5. Press the *Close* button.

6. Expand the *Validation Rules* on the *Rules and Triggers* tab to display the details.



This is a *List of Values* rule. Notice that this is a similar rule as the Simple Logic Check rule specified for the Department field but expressed in a different way.

7. Display the *Details* tab. Compare the Rule Definition and the Actions with iiiDeptCode.

🕎 Repository 🛛 🚓 Outlin	e 📜 Details 🔶 Favorites 🖕 💷
Description	
	1 2
Sequence Rule Definition	1 ÷
Kule Definition	Value
*BLANK	value
DLAINK	
Validation Usage	
When Inserting	Always apply rule (ADD)
When Updating	Always apply rule (CHG) -
When Deleting	Never Apply Rule -
Actions	
In The List	Set field in error (ERROR)
Not In The List	Evaluate next rule (NEXT)
Error Message	
Message Number	DCM1077
Message File	DC@M01
Message Text	
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

- 8. Save and close the iiiSurname field.
- 9. Close the iiiDeptCode field.

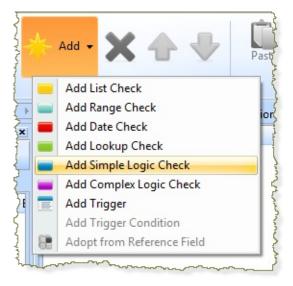
Step 3. Create a Rule for the iiiSalary Field

In this step you will create a simple logic rule to ensure that the value of the iiiSalary field is always greater than 0.

- 1. Open the **iiiSalary** field in the Editor.
- 2. Select the Rules and Triggers tab.

There are no rules defined for this field.

3. Click the Add button on the *Home* ribbon and select the Add Simple Logic Check option.



A new simple logic validation rule is displayed.

4. In the *Details* tab and the *Rule Definitions*, create the list check rule as follows:

Rule Definitions:

Description	Salary must be greater than 0
Sequence	1
When inserting	Always apply rule (ADD)
When updating	Always apply rule (CHG)
When deleting	Never apply rule
Value	#iiiSalary *gt 0

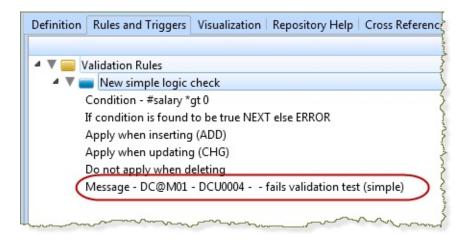
Your *Details* tab should now look like this:



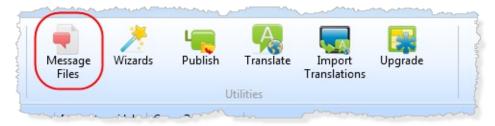
5. Review the *Actions* settings, ensure the rule is created as follows:

If the condition is found to be true	
If the condition is found to be false	Set field in error (ERROR)

The Error message file and message number have been created automatically. Note that the error message text for this message is not very helpful:



6. To find a more specific error message, on the *Home* ribbon, select the *Message Files*... option.



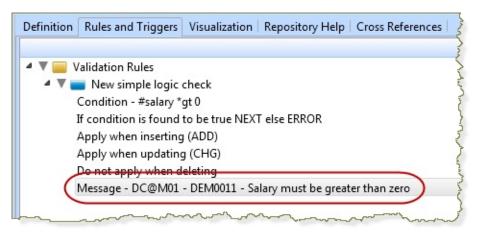
7. In the *Message File Maintenance* dialog type **greater** in the 1st level text contains field and click *Find*.

Note that message *DEM0011* is appropriate for the rule.

1st level te	earch Criteria ext contains: ext contains:	greater		Message file: DC@M01 Language:	
ID from:	to:		Find	English	
Message ID/	Text		No. of Mes	sages: 18	
ADW0147	First value for keyword &1 must not be greater than				
CMP0292					
DCM0910	If greater than go to label Effective Start date cannot be Greater than Effectiv =				
DCM1082	&1 requires value greater than zero.				
DCM1099	Overflow number cannot be greater than document length				
DCM1213	File &1 cannot be a "HST" - Aggregate record length				
DCM1273	** Push but	ton and Drop	down fields can	not be greate	
DEM0011	Salary must	be greater th	an zero		
LMD0093	Option greater than 1 not valid on the create model line.				
4					

8. Close the dialog and specify **DEM0011** as the Message number. Notice the

new message text for the field.



For your own application, we recommend that you use a message from your own message file instead of manually entering the message text for the field.

Do not add your messages to the shipped LANSA message file, DC@M01.

9. Save and close the field.

Summary

Important Observations

- Field validation rules are applied when file operations are performed. Because you have not yet created any files, you cannot test the field validation rules in this exercise.
- There may be many ways of performing the same check.

Tips and Techniques

- Remember that most business rules should be defined for a file, not a field.
- If you us messages with the variables (e.g. &1),validation error handling will substitute the field label for this variable.

What I Should Know

- How to define rules for a field in the LANSA Repository.
- How to control the action performed when a rule is checked.
- How to define error messages for rules.
- How to adopt rules from a reference field.

REP004 - System and Multilingual Variables

System variables are used to store commonly used and often variable pieces of information. They are global variables that are used across all LANSA partitions.

Multilingual variables are specific to a partition and are used to store commonly used text in several languages. These variables can be used as field values, in validation rules or as command parameters. Multilingual variables are a part of LANSA's multilingual application support.

Objectives

- To highlight some of the system variables in LANSA and how they are used.
- To create a system variable which will automatically generate the next available number for determining Employee codes.
- Optional: To review how multilingual variables are used in LANSA.

To achieve these objectives, you will complete the following:

Step 1. Review an Existing System Variable

Step 2. Create a System Variable

Step 3. Assign the System Variable as a Default Value

Step 4. Test System Variable using form iiiTestFields

Step 5. Review the System Variable (Optional)

Step 6. Review an Existing Multilingual Variable (Optional)

Summary

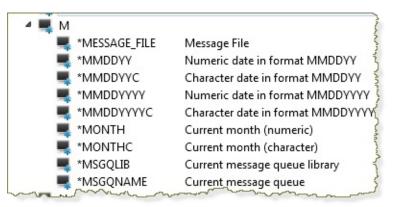
Before you Begin

In order to complete this tutorial, you must have completed the previous tutorials.

Step 1. Review an Existing System Variable

In this step you will review an existing system variable.

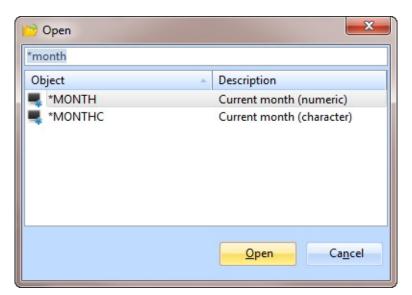
- 1. In the Repository tab, expand the Resources group.
- 2. Expand System Variables.
- 3. Expand the **M** group (if the system variables are grouped alphabetically) to show all system variables starting with M.



4. Locate and open the ***MONTH** system variable in the *Editor*. You will see a "Read Only" dialog as you open it.

4 🜉 Details	
Name	*MONTH
Description	Current month (numeric)
Derivation	Static
Data type	Numeric
Length	2
Decimals	0
3GL program	M@SYSVAR

Note that, alternatively, you can open the *MONTH system variable by typing **Ctrl** + **O**, and specifying *MONTH:



5. Review the system variable definition.

This system variable returns the current month. It uses the LANSA supplied program M@SYSVAR to set the value. Do NOT change the system variable.

6. Close the system variable.

Step 2. Create a System Variable

In this step you will create a system variable which increments a number **nn** (2 digits) by one and returns it as an alphanumeric value.

To create this system variable you use a LANSA shipped evaluation program M@SYSNUM. The names of system variables using this program must start with *AUTOALP.

- 1. Press the *New* button on the toolbar and select *System Variable* from the drop down list. The *New system variable* dialog is displayed.
- 2. Enter the following characteristics for the system variable:

Name	*AUTOALP02iiiNUM (where iii are your initials).
Description	Next Available Number
Method of derivation	_l Dynamic
Data type	Alphanumeric
Length	2
Program Type	3GL program
Program name	M@SYSNUM

Name	*AUTOALP02IIINU	M	Create
Description	Next Available Nu	mber	
Derivation	Dynamic	.	Cancel
Data type	Numeric	(*)	
Length	2 ‡		
Decimals	0 \$		
Program Type	3GL program		
Program Name	M@SYSNUM		

3. Press the *Create* button.

4. The system variable has been created.

The value of the system variable will be set automatically when it is accessed in the next step.

Step 3. Assign the System Variable as a Default Value

In this step you will assign the system variable you created as the default value of the iiiEmployNumber field. By using this system variable, the value of the field will be incremented by 1 each time the field is used.

- 1. Open the **iiiEmployNumber** field in the Editor.
- Change the Default value to use the new system variable you created,
 *AUTOALP02iiiNUM. You can do this by entering *AUTO and using the prompt button.

🛧 Outline 🛛 📜 Deta		Definition Rules and Triggers	Visualization Repository Help Cross
Name	iiiEmployNumber	4 📑 Details	}
Identifier	IIIEMPLOY	Name	iiiEmployNumber
Field Type	Alpha -	Identifier	IIIEMPLOY
Field Length	2 *	Туре	Alphanumeric
Decimals	÷	Length Decimals	2
Default Value	*AUTOALP02IIINUM ···)	Default Value	*AUTOALP02IIINUM
Reference Field		Reference Field	Ş
Descriptions		Description	Employee ID

The find dialog will display objects with names like *AUTO.

System Variables	A Multili	ngual Variables	
ike name	AUTO		
ike description			
			Find
Name		Description	
💐 *AUTOALP02IIINUI	M	Next Available Number	
AUTONUM15EVE	NTNUM	Event number	

- 3. Select your ***AUTOALP02iiiNUM** system variable and press *OK*.
- 4. Save and close the field.
- 5. This type of system variable stores the "next number" in a data area (in this case a data area named IIINUM, which is defined as the last part of the system variable name).

The data area is read with a lock, incremented, returns the value to your form and then updates the data area (and releases the lock).

If the data area does not already exist, it is created when first referenced. It is stored in the *partition module library*.

Step 4. Test System Variable using form iiiTestFields

In this step you will test the changes made to the iiiEmployNumber field using your iiiTestFields form.

- 1. To test your change to the default value of the iiiEmployNumber field, you need to force the recompile of the test form.
 - a. On the *Favorites / Last Opened* tab, locate your iiiTestFields form.
 - b. Right click on **iiiTestFields** and select the *Compile* option from the context menu.
 - c. When the *Compile options* window appears, deselect the *Compile only if necessary* option.

🐣 Compile options		
	Component Compile Options	
Components (1)	Compile only if necessary	
	Debug enabled	
	Keep generated source	
Use Default Settings	OK Cancel	

d. Press *OK* to submit the compile.

Notice that the status of the submitted compile appears in the *Compile* window at the bottom of the Editor.

2. Once your form has compiled, execute it.

Notice the default value for the Employee Number which appears when the form first displays. This value was assigned by accessing the system variable and determining the next available number.

Employee ID	01

- 3. Close the form.
- 4. Execute the form again.

Note the value in the Employee Number field has been incremented by one.

The system variable automatically updates to the next available number each time it is accessed (by your application or any other application in any partition in the system).

Employee ID	02

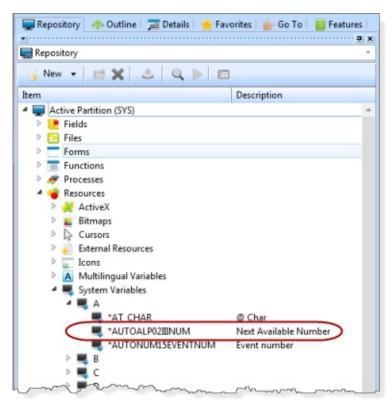
If 50 people were executing your test form, each person would see a different default value.

When the Employee code reaches 99, it will reset to 1.

Note: You normally use this type of system variable to generate unique numbers, such as the next available order number.

Step 5. Review the System Variable (Optional)

- To do this step, you will need more than one partition in your LANSA system as you will be looking at how the system variable you just created is available across partitions.
- 1. If you have more than one partition in your Visual LANSA system, logon to Visual LANSA in another partition.
- 2. Review the list of fields in the partition. You will not see any of the fields you have created in these tutorials.
- 3. Review the list of system variables. You will see the new system variable you created because these variables are accessible by all partitions in the system.



4. Close Visual LANSA.

Step 6. Review an Existing Multilingual Variable (Optional)

To do this step, your partition must be set up to support development of multilingual applications.

- 1. In this step you will create three multilingual variables with English and French values.
 - a. From the New button on the toolbar select Multilingual Variable

A New Multilingual	Variable	×
Name	*МТХТ	Create
Maximum Length	78 ‡	Cancel
Description	•	Open in editor
		Close

- b. Ensure the Close option is not selected initially, but do open each multilingual variable in the editor.
- c. Create the following three variables:

Name	Description
*MTXTiiiDETAILS	Details
*MTXTiiiAddress	Address
*MTXTiiiSave	Save

Ensure you close the dialog when creating the third variable.

- 2. In the Editor, switch to the Details multilingual variable.
 - a. Double click on the Multilingual Details to open the Details tab on the left.
 - b. Change the French value to **Détails**. You can copy this value from the tutorial in the online guide.
 - c. Save and close *MTXTiiiDETAILS
 - d. Switch to the Address variable, and change the French value to **Adresse**.
 - e. Save and close *MTXTiiiADDRESS

- f. Switch to the Find variable and change the French value to **Trouver**
- g. Close and save *MTXTiiiFIND.
- 3. Create a new form, iiiMultilingual Test.

Replace all from code with the following:

FUNCTION OPTIONS(*DIRECT) BEGIN COM ROLE(*EXTENDS #PRIM FORM) CLIENTHEIGHT(322) CLIENTWIDTH(391) HEIGHT(360) LEFT(571) TOP(200) WIDTH(407) DEFINE COM CLASS(#PRIM TAB) NAME(#TAB 1) BOTTOMLAYOUTPRIORITY(4) COMPONENTVERSION(1) DISPLAYPOSITION(1) HEIGHT(241) LEFT(8) LEFTLAYOUTPRIORITY(1) PARENT(#COM OWNER) RIGHTLAYOUTPRIORITY(2) TABPOSITION(1) TOP(8) TOPLAYOUTPRIORITY(3) WIDTH(369) DEFINE COM CLASS(#PRIM STBR) NAME(#STBR 1) DISPLAYPOSITION(2) HEIGHT(24) LEFT(0) MESSAGEPOSITION(1) PARENT(#COM OWNER) TABPOSITION(2) TABSTOP(False) TOP(298) WIDTH(391) DEFINE COM CLASS(#PRIM PHBN) NAME(#FIND) DISPLAYPOSITION(3) LEFT(18) PARENT(#COM OWNER) TABPOSITION(3) TOP(264) DEFINE COM CLASS(#PRIM TBSH) NAME(#TBSH 1) DISPLAYPOSITION(1) HEIGHT(213) DISPLAYPOSITION(1) HEIGHT(21) LEFT(14) PARENT(#TAB 1) TABPOSITION(1) TOP(16) USEPICKLIST(False) WIDTH(237) DEFINE COM CLASS(#PRIM TBSH) NAME(#TBSH 2) DISPLAYPOSITION(2) HEIGHT(213) DISPLAYPOSITION(1) HEIGHT(21) LEFT(14) PARENT(#TAB 1) TABPOSITION(1) TOP(16) USEPICKLIST(False) WIDTH(237) DEFINE_COM CLASS(#SURNAME.Visual) NAME(#SURNAME) DISPLAYPOSITION(2) LEFT(16) PARENT(#TBSH 1) TABPOSITION(2) TOP(51) WIDTH(321) DEFINE COM CLASS(#GIVENAME.Visual) NAME(#GIVENAME) CAPTION(*MTXTIIIDETAILS) DISPLAYPOSITION(3) LEFT(19) PARENT(#TBSH 1) TABPOSITION(3) TOP(86) WIDTH(326) DEFINE COM CLASS(#ADDRESS1.Visual) NAME(#ADDRESS1) COMPONENTVERSION(1) DISPLAYPOSITION(1) HEIGHT(21) PARENT(#TBSH 2) TABPOSITION(1) TOP(21) USEPICKLIST(False)

WIDTH(350)

DEFINE_COM CLASS(#ADDRESS2.Visual) NAME(#ADDRESS2) COMPONENTVERSION(1) DISPLAYPOSITION(2) HEIGHT(21) PARENT(#TBSH_2) TABPOSITION(2) TOP(43) USEPICKLIST(False) WIDTH(350) DEFINE_COM CLASS(#ADDRESS3.Visual) NAME(#ADDRESS3)

COMPONENTVERSION(1) DISPLAYPOSITION(3) HEIGHT(21) PARENT(#TBSH_2) TABPOSITION(3) TOP(65) USEPICKLIST(False) WIDTH(350)

DEFINE_COM CLASS(#POSTCODE.Visual) NAME(#POSTCODE) COMPONENTVERSION(1) DISPLAYPOSITION(4) HEIGHT(21) PARENT(#TBSH_2) TABPOSITION(4) TOP(87) USEPICKLIST(False) WIDTH(249)

EVTROUTINE handling(#com_owner.Initialize) SET #com_owner caption(*component_desc) ENDROUTINE

EVTROUTINE HANDLING(#FIND.Click) fetch *all FROM_FILE(PSLMST) with_key(#empno) ENDROUTINE

END_COM

- 4. Save your form.
- 5. Switch to the *Design* tab.
 - a. Select the push button. On the *Details* tab changes its Caption to *MTXTiiiFIND, by selecting it using the ellipsis button and *Repository Find* dialog.
 - b. Select the first Tab Sheet (contains Employee Number). Click first on the tab, and then in the middle of the sheet to select it.

H 1		4	Details
-			
- 8			Details
perties Events Met	hods		
Caption	*MTXTIIDETAILS		Employee Number ABCDE
Cursor	*NULL		
DisplayPosition	1		Employee Surname aAbBcCdDeEfFgGhHitjJ
DragStyle	None		
EnableChildren	False		Employee Given Name(s) aAbBcCdDeEfFgGhHiljJ
Enabled	True		
Height	213		
Hint			
HintPopup	*NULL		1
HintShow	True		
HintShowOfParent	True		
HintTitle			
Image	*NULL		Find
LayoutManager	*NULL		
Left			aAbBcCdDeEfFgGhHiljJkKlLmMnNoOpPgQrRsStTuUvVwWxXyY -

- c. Select the second Tab Sheet (contains Address Line 1) and set its Caption to *MTXTiiiADDRESS.
- 5. Compile the form.
- 6. Execute form iiiMultiLingualTest from the development environment. This is running the form using the English language and it should look like the following:

Demo Multillingual	
Details Address	
Employee Number	A0090
Employee Sumame	Bloggsxxx
Employee Given Name(s)	Fred
Find	
	al.

7. From the Windows start menu, select *LANSA / Exec Form*:

Execute Form on workstation					
Form Name	IIIMULTI				
Language	FRA				
Partition Identifier	TRN				
LANSA User	JIVORY13				
Database User	USERID				
Database Password					
Database Name	LX2JOHNI				
Database Type	MSSQLS				
Default Printer	LPT1				
Debug	N				
Debug Host	johnivory-pc:51237				
Trace	N				
Max Trace Lines	20000				
Trace level	4				
Trace Categories	ALL				
Heap Validation	N				
Render Type	W				
Graphics Processing	Н				
OK Cancel	Help Parameter Help				

- a. Change Language to FRA.
- b. Ensure Render Type = W

Note: The Form Name will be pre-filled with the Identifier for your form, iiiMultiLingualTest, which was the last form run in the development environment.

c. Click OK to run the form, using French.

Enter an employee number such as A0090, A1004 or A1005, and click Trouver.

Your form should look like the following:

Adresse ligne 1	70 King Street	
Adresse ligne 2	Newtown	
Adresse ligne 3	NSW	
Code postal	2220	

Note:

- The field descriptions in French are defined in the Repository for each field
- The Caption on the Find button and the Tab Sheets are defined by the multilingual variables.

Summary

Important Observations

- System variables can be used throughout the repository as default values, in validation rules, etc.
- LANSA ships many default system variables. You can also define your own system variables.
- System variables are NOT partition specific and changing or deleting a system variable affects all partitions in that installation of LANSA.
- Dynamic system variables can have different values each time the variable is accessed.
- Static variables are assigned once when the application is started.
- System variables call a program which returns a single value.
- Multilingual data is stored as keyed data and is not translated.
- All multilingual variables must begin with *MTXT.

Tips and Techniques

• When defining a new partition (or modifying an existing partition) to support multilingual applications, a "default" language and one other language must be defined. A multilingual partition must have two or more languages defined.

What I Should Know

- How to create a system variable in Visual LANSA.
- Some of the special system variables supported by LANSA.
- Where a LANSA system variable can be used.
- How multilingual variables are used in Visual LANSA.

REP005 - Creating Files

A LANSA file may also be referred to as a *database table*. Fields are added to the file to define the record format (key columns and columns). A LANSA field can also be described as a *database column*. Once a file is compiled, the records in it can be retrieved, added, updated and deleted.

Objectives

- To create two file definitions. You will create a Department Table and an Employee Table. These tables are similar to the ones used in the DirectX Examples application.
- To review the database file attributes which can be defined in the LANSA Repository.
- The file definitions will be as follows:

Department	File iiiDepartments	Employee File	iiiEmployees

iiiDeptCode*	iiiEmployNumber*
iiiDeptDescription	iiiDeptCode
	iiiSurname
	iiiGivenName
	iiiSalary
	iiiStartDate
	iiiEmployNotes

* indicates the fields used as the file key

Note: Files can be created manually or by copying an existing file definition in the LANSA Repository. In this lesson, you will manually create the file definitions so that you can see all the steps involved. If you copy an existing LANSA file definition, the fields, rules, and all other related database information about the file can be copied.

To achieve these objectives you will complete the following:

Step 1. Create File Definition

Step 2. Compile the File

Step 3. Create Department Maintenance Form

Step 4. Execute Department Maintenance Form

Step 5. Create the Employee File

Step 6. Create Employee Maintenance Form

Step 7. Execute Employee Maintenance Form

Step 8. Database Attributes

Summary

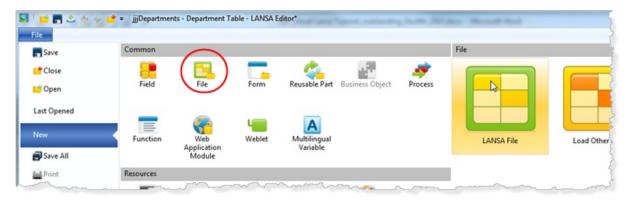
Before you Begin

In order to complete this tutorial, you should have completed the previous tutorials.

Step 1. Create File Definition

In this step you will create iiiDepartments Department file.

1. Select the *New* button in the *File* menu and select *LANSA File*.



a. Enter the following characteristics for the file:

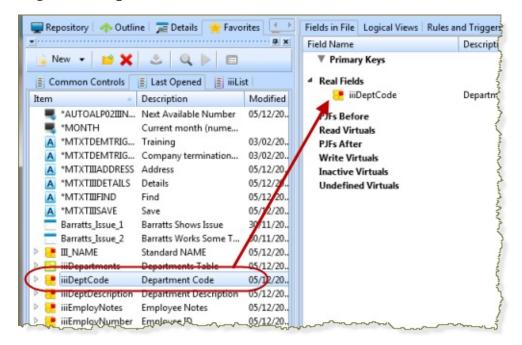
File name	<pre>iiiDepartments (where iii=your initials)</pre>
Description	Departments Table
Enabled for RDMLX	Yes

🔁 New File		×
Name	iiiDepartments	Create
Description	Departments Table	Cancel
Library	D13TRNLIB	
Identifier	IIIDEPARTM	Open in editor
Enabled For RDMLX		

- b. Select the option to Open in editor.
- c. Press the *Create* button.
- 2. You will now start adding fields to the file.
 - a. Make sure the Fields in File tab is displayed.

Fields in File	Logical Views	Rules ar	nd Triggers	Access Routes
Field Name			Description	ר ל
V Primary	y Keys			<
▼ Real Fie	elds			<
PJFs Befo	re			Į
Read Virt	uals			(
PJFs After	r			1
Write Virt	tuals			2
Inactive V	/irtuals			ŝ
Undefine	d Virtuals			

- b. Display the iiiLIST tab which you created in exercise REP001- Step 6. Create a Dynamic List for Your Fields .
- c. Drag the **iiiDeptCode** field to the file under Real Fields.



d. With the iiiDeptCode field selected in the file, click on the Key Field button on the *Home* ribbon, to make it the key field of the file.



(You can also drag the field directly under Primary keys to make it a key

field.)

- e. Double-click iiiDeptCode to display the Details tab.
- f. Notice the *Key position* is **1** to indicate that iiiDeptCode is the first key field in the file.

	Details 🔹					
	IIIFILDEPT					
	Field name		IIIDEPT			
	Description		Department Code			
	Туре		Alphanumeric			
	Length		3			
	Decimals					
	Sequence		1			
	Key position		1			
	Default value		*BLANKS			
	Allocated length					
	Input attribu	ites	FE - Field exit key required			
		_				
	•	0.11		1 A		
Re Re	pository	p Outli	ne 🛛 📜 Details 📩 Favorites	· 뮌 ×		
Field N	eld Name iiiDeptC		Code			
Sequen	ice	1		¢		
Key Pos	sition			¢		
Allocat	ocated Length			\$		
Field D	efinition					
Descrip	tion	Departr	ment Code			
June	Tune		upparies	m		

g. Add the iiiDeptDescription field to the file using drag-and-drop. Your file will appear as follows:

rields in Fi	Ile Logical Views	Rules and Triggers	Access Routes	Batch Control	File Attributes	Relationships	Cross Reference	es Virtua	Derivation
Field Nam	ne	Descriptio	n			Ref. Field	Туре	Length	Decimals
▼ Prin	nary Keys								
Real Fi	ields								
	iiiDeptCode	Departme	nt Code				Alpha	4	
	iiiDeptDescription	Departme	nt Description				Alpha	20	
PJFs B	efore								
Read \	/irtuals								
PJFs A	fter								
Wote	Virtuals		~~~~~	~ m	~~~~~		-		~~~~
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		and the		- · · · · · · · · · · · · · · · · · · ·			~~~	~~~

# Step 2. Compile the File

Now that the definition is complete, the file must be compiled to be operational. 1. Display the *Compile options* dialog to submit your file for compile.



In this way you can view the compile settings before submitting the job. The *File Compile Options* dialog is displayed.

🗳 Compile options	×
	File Compile Options
🔁 Files (1)	Compile only if necessary
	<ul> <li>Rebuild table</li> <li>Rebuild indexes and views</li> </ul>
	Rebuild OAMs
	Strip debug information
	Keep generated source
	Keep saved data (DAT file)
	Drop existing tables/indexes
	Save table data
	Reload table data
Use Default Settings	OK Cancel

2. Press the *OK* button to submit the compile using the defaults shown.

**Note:** As a new file definition, the option *Compile only if necessary*, will build the table and any indexes or views, and create and compile an OAM.

3. A compilation status will be displayed in the Compile tab.

Make sure that the compile completed successfully.

2	Job Status	Description	Results	-
	Completed	iiiDepartments - Departments Table	Compiled 1 of 1	$\mathbf{)}$
tir.	Completed	iiiMultiLingualTest - Demo Multillingual	Compiled 1 of 1	_
×	Completed	iiiMultiLingualTest - Demo Multillingual	Compiled 1 of 1	
^	Completed	iiiMultiLingualTest - Demo Multillingual	Compiled 1 of 1	
	4			
Assist	ant  📩 Compile 🐗	Check Out   📕 Propagation   🛼 Text Search   🍕	🕜 Help	
			LANSA13 TRN	jivory1

4. Close the file.

# **Step 3. Create Department Maintenance Form**

In this step you will create a test form which will allow you to add, change, delete and display the data in the department file you have just created.

- 1. Create the Department Maintenance form.
  - a. Create a new form with the following characteristics:

Name	iiiMaintDepartment	(where iii=your initials)
Description	Department Maintena	nce

Your *New Form dialog* should look like this:

New Form		x
Name	iiiMaintDepartment	Create
Description	Department Maintenance	Cancel
Framework	Personnel & Payroll (HUMAN RESOURCES)	cuncer
Group		
Identifier	IIIMAINT	
Enabled For RDMLX		

The form will open in the Editor.

b. In order to use a template, you need to know the *Identifier* for the file iiiDepartments. Templates do not recognize long names.

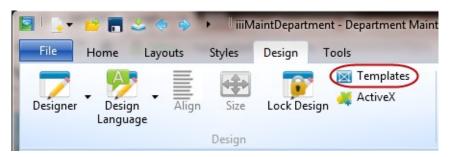
On the *Favorites* tab, expand the width of the *Last Opened* tab. If necessary, use the right mouse menu on a column heading to show the *Identifier* column.

Make a note of the file iiiDepartment's Identifier. For example iiiDEPARTM.

00	10113_1330C_2	bunatts works some nines	DAULAT_T	Dana an	
🕨 📜 🎹	NAME	Standard NAME	III NAME		Á
🔍 🖪 iiiD	epartments	Departments Table	<b>IIIDEPARTM</b>	Compiled	D
🖻 📒 iiiD	eptCode	Department Code	IIIDEPTCO		R

c. With your new form selected in the editor, click the Templates button on

the Design ribbon



Press the *Template* button in the *Design* ribbon to display the *Choose Template* dialog.

d. Select the *VL_BASEMNT Visual LANSA flat file maintenance* template and press the *Execute* button.

Choose Temp	late	x
VL_BASEBOB	Visual LANSA Business Object Browser	*
VL_BASEBOD	Visual LANSA Business Object Detailer	
VL_BASEBOF	Visual LANSA Business Object Filter	
VL_BASEBR1	Visual LANSA advanced parent browser	
VL_BASEBR2	Visual LANSA advanced tabbed detail	
VL_BASEMNT	Visual LANSA flat file maintenance	
VL_BASETAB	Visual LANSA tab folder with four sheets	
VL_BASETRE	Visual LANSA simple tree list	
VL_BASEWRK	Visual LANSA work with object	
VL_BBMNUBR	Visual LANSA: Menu bar building block	
VL_BBSTSBR	Visual LANSA: Status bar building block	
VL_BBWEBRT	Visual LANSA: Webroutine Building Block	
VL_FBASBOB	Visual LANSA Business Object Browser	
VL_FBASBOD	Visual LANSA Business Object Detailer	
VL_FBASBOF	Visual LANSA Business Object Filter	
VL_FBASBR1	Visual LANSA advanced parent browser	
VL_FBASBR2	Visual LANSA advanced tabbed detail	
VL_FBASMNT	Visual LANSA flat file maintenance	-
Execute	Cancel Display All	

e. Answer the template questions as shown in the table below. (Remember to replace iii with your 3 character identifier.)

Template/Wizard Question	ANSWER
Supply a word that describes WHAT this data entry program works with	Department
Enter the name of the PHYSICAL file to be used by this template	iiiDEPARTM

How do you want to display the fields?	FF
Fields to appear on display	Select both fields

The template will display Identifiers for your fields.

f. Press the *Finish* button once you have answered all questions.

The template replaces the default code in your new form. The form is complete and ready for compilation. It is not necessary to change the source RDMLX.

g. Select the *Design* view. Reposition the buttons (select as a group and use Ctrl+Cursor keys.) and resize the form.

Department		- 0 ×
Department Code Department Description	aAbBcCdDeEfFgGhHiliji	
<u>F</u> etch	Insert Update	Delete
aAbBcCdDeEfFgGhHiliJkKlLmN	InNoOpPqQrRsStTuUvVwWxXyYzZ	•

- 2. Next, compile your form.
  - a. Select the *Home* ribbon if necessary.
  - b. Press the *Compile* **Compile** button. Compile will automatically save the form.
  - c. Review the *Compile* tab to ensure that the compile completed successfully.

### **Step 4. Execute Department Maintenance Form**

In this step you will execute the Department Maintenance form and add some data to the Department file.

1. Execute form **iiiMaintDepartment** by clicking *Runtime / Execute* on the *Home* ribbon.

Department Maintenar	nce		
Department Code Department Description <u>F</u> etch	ADM Administration Insert Update	<u>D</u> elete	*** ***

2. Insert the following data to the file:

#### **Department Description**

ADM	Administration
LGL	Legal
MRK	Marketing
TRN	Company Training
TRV	Business Travel

Note that if you try to insert a blank Department and Description, you will see that the messages caused by the field level validation rules created in REP003 - Validation Rules.

- 3. Close the form.
- 4. Close the form in the Editor.

# **Step 5. Create the Employee File**

In this step you will create an Employee file using the skills you have just learned in the previous steps. You will use the multi-add fields feature to add multiple fields to the newly created file.

- 1. Create the Employee file.
  - a. Use *New / File / LANSA File*. Enter the following characteristics for the file:

File name	<pre>iiiEmployees (where iii=your initials)</pre>
-----------	---------------------------------------------------

Description Employee File

Enabled for RDMLX Yes

New File				
Name	iiiEmployees		Create	
Description	Employee File		Cancel	
Library	D13TRNLIB			
Identifier	IIIEMPLOYE		Open in editor	
Enabled For RDMLX		-		

- b. Select the option to *Open in editor*.
- c. Press the *Create* button.
- 2. On the Home ribbon, select *Add / Multi-add Fields*.



The Add Fields to the File window is displayed.

Add Fields to th	erile						
Field Name	Description	Reference Field	Туре	Length	Decs	Key	Virtual
					OK		Cancel

**Note:** When typing a field name, the autocomplete feature will attempt to identify an existing repository field name as you type in the characters.

3. Add the fields iiiEmployNumber, iiiDeptCode, iiiSurname, iiiGivenName, iiiSalary,iiiStartDate and iiiEmployNotes to the file.

You should only need to type 4 or 5 characters and autocomplete will match with the right field.

Specify that iiiEmployNumber is the key field in the file by entering the value 1 in the Key column.

Field Name	Description	Reference Field	Type	Length	Decs	Key	Virtual
iiiEmployNumber	Employee ID		Alpha	2		1 ‡	Г
iiiDeptCode	Department Code		Alpha	4			
iiiSumame	Surname	III_NAME	Alpha	25			
iiiGivenName	Given Name	III_NAME	Alpha	25			
iiiSalary	Monthly Salary		Packed	11	2		
iiiStartDate	Start Date		DateTime	26			
iiiEmployNotes	Employee Notes		String	512			

4. Once you have added all the fields to the file, click OK. Your definition should appear like this:

ield Name	Description	Ref. Field	Type	Length	Decimals
Primary Keys					
🔑 iiiEmployNumber	Employee ID		Alpha	2	
Real Fields					
🔑 iiiEmployNumber	Employee ID		Alpha	2	
📜 iiideptcode	Department Code		Alpha	4	
📑 iiisumame	Surname	III_NAME	Alpha	25	
📒 iiigivenname	Given Name	III_NAME	Alpha	25	
: iiisalary	Monthly Salary		Packed	11	2
🔻 📑 iiistartdate	Start Date		DateTime	26	
iiiemploynotes	Employee Notes		String	512	

- 5. Compile the file.
- 6. Close the file definition.

## **Step 6. Create Employee Maintenance Form**

In this step you will create a test form which you can use to add, change, delete and display the data in the employee file you have just created.

- 1. Create the Employee Maintenance form.
  - a. The template requires the file's *Identifier*. Look up the *Identifier* for the file on the *Favorites / Last Opened* tab. For example IIIEMPLOYE.
  - b. Create a form with the following characteristics:

Name	iiiMaintEmployee (where iii=your initials)
Description	Employee Maintenance
RDMLX Enabled	Yes

- c. Once the Editor has opened, press the *Template* button is on the *Design* ribbon.
- d. In the *Choose Template* dialog, select the *VL_BASEMNT Visual LANSA flat file maintenance* template. Press the *Execute* button.
- e. Answer the template questions as shown in the table below. (Remember to replace iii with your 3 character identifier.)

Template/Wizard Question	ANSWER
Supply a word that describes WHAT this data entry program works with	Employee
Enter the name of the PHYSICAL file to be used by this template	IIIEMPLOYE
How do you want to display the fields?	FF
Fields to appear on display	Select ALL fields.

Your form is now created and ready for compilation. It is NOT necessary to edit the source.

f. Adjust the width of the fields on the form.

You may find this quickest if you select a field and adjust its *Width* property by guessing a smaller value. Then drag the field making it wider if necessary.

Employee	
Employee ID	AB
Department Code	ABCD
Sumame	aAbBcCdDeEfFgGhHiljJkKILm
Given Name	aAbBcCdDeEfFgGhHiljlkKlLm
Monthly Salary	123,456,789.12
Start Date	01/01/1900
Employee Notes	aAbBcCdDeEIFgGhHiljikKILmMnNoOpPqQiRsStTu UVVwVxVyYzZaAbBcCdDeEIFgGhHiljikKILmMnN oOpPqQiRsStTuUVVwVXVyYzZaAbBcCdDeEIFgG hHiljikKILmMnNoOpPqQiRsStTuU WeWXVyYzZaAbBcCdDeEIFgGhHiljikKILmMnNo OpPqQiRsStTuUVVwVxVyYzZ
Factor in the	
·	
aADBCCdDeEIFgGhHipkKiLmM	nNoDpPqQrRsStTuUvVwWxXyYzZ

2. Compile your form.

### **Step 7. Execute Employee Maintenance Form**

In this step you will execute the form and enter some data into the Employee file.

1. Execute your application.

Employee ID	03
Department Code	ADM
Sumame	Brown
Given Name	Veronica
Monthly Salary	24,500.00
Start Date	07/12/2012
Employee Notes	Lorem ipsum dolor sit amet, cu vel nostro impetus, vis diam placerat invidunt ei. Modo audire placerat mel in. Cum omittantur neglegentur an, usu in nostrum signiferumque. Oblique neglegentur nec eu, munere dissentiet nam in. In alia erant altera eum, ut dicit admodum ponderum mea, quo ei solum graecis.

2. *Insert* the following data to your file. (You can use the data in the table below or create your own.)

Employee Number	Department	Surname	First Name	Monthly Salary	Start Date	Notes
01	ADM	Brown	Veronica	5000.00	01/01/1989	Et harur dereud facilis e er expec distinct. Nam lib a tempo cum sol nobis eligend optio

						comque nihil qu a imped anim id quod maxim placeat facer possim omnis e voluptas assumer est, omr dolor repellen
02	ADM	Smith	Ben	3000.00	23/04/2001	same as employ( 01
03	LGL	Jones	Dan	2900.00	10/12/2000	same as employ( 01
04	TRV	Turner	Jennifer	6500.00	05/02/1990	same as employ: 01

3. When you have entered all the employee details, exit the form.

4. Close the form in the Editor.

# **Step 8. Database Attributes**

In this step you will simply review the file database attributes to highlight what information is accessible. You will not make any changes.

- 1. Open the **iiiEmployees** Employee file (if it is not already open).
- 2. Select the *File Attributes* tab. Double click a setting such as *I/O Module Library* so that the *Details* tab is shown..

Name	iiiEmployees	4 File Settings		
Identifier	IIIEMPLOYE	Name	iiiEmployees	
Library	D13TRNLIB	Identifier	<b>IIIEMPLOYE</b>	
Record Format Name	IIIEMPLOVE	File Type File Library	LANSA file D13TRNLIB	
/O Module Library		Record Format Name	MEMPLOYE	
	Same library as file (F)	1/O Module Library	Same library as file (F)	
Alt. Collating Table		File Uses SQL On IBM i	No	
Options		Alt. Collating Table		
Enabled For RDML	x	Enabled For RDMLX	Yes	
		Enabled For Long Names	Yes	
Enabled For Long N	Vames	Share Data Path	No	
Share .		Secure From File Overrides	Yes	
Secure		Strip Debug On Compile	Yes	
Strip Debug		Suppress IOM0034 Message	No	
		Ignore Decimal Data Error	No	
Suppress IOM0034		Create I/O Module When Compiling	Yes	
Ignore Decimal Dat	ta Error	Create Batch Control Header Records	No	
IOM Required		IBM i High Speed Table	No	
Create Batch Contr		Auto Relative Record Number Generati		
		Create Relative Record Number Column		
BM i High Speed T	able	Convert Special Characters In Field Nan		
Auto RRN Generati	on	Enable Commitment Control Auto Commit	No	
Create RRNO colur	790	CRTPF And CHGPF Parameters	No SIZE(10000 2000 3) LVLCHK("VES)	
		Readonly Access	No	
	aracters In Field Names	Enable Database Triggers	No	
Commitment Control	Options	Database Trigger Program	140	
Commitment Cont	trol			
Auto Commit		Descriptions		
Database Trigger Prog	ram	English	Employee File	
		French	Employee File	
Enable Database Tr	nggers	✓ ▼ Supported Database Products		
		Adaptive Server Anywhere (for all Int	el Platforms)	
CRTPF And CHGPF Pa	rameters			
SIZE(10000 2000	3) LVLCHK("YES)	Job Status Description	Results	

3. If you select a setting in the *Details* tab, you can use the *F1 help* to review the detailed information about file settings in the *online guide*.

Notice that you are also able to specify the multilingual descriptions for the file name if the partition is multilingual.

Do not make any changes to the file attributes.

4. Close the file.

## Summary

#### **Important Observations**

- Notice that the field level validation rules you created in exercise REP003 Validation Rules are enforced when you maintain the files. Field level validations are enforced in every file in which the field is used. However, remember most validation rules should be defined at file level.
- Key sequence is used to define the order of the keys in the file. The physical sequence of fields and the key sequence are independent.
- Access to files is handled by the Object Access Modules (OAM). Every file used by LANSA has an OAM which contains the repository features of the file and its fields. The OAM is executed whenever the file is accessed by a program.
- When you make a change to a field used in a file or to the file definition itself, you need to recompile the file. You do not need to recompile your programs unless the change affects the visual display.

# **Tips and Techniques**

- Use the multi-add fields feature as a faster way to add fields to a file. You can include fields that do not exist in the repository and these fields will be created when you exit the multi-add window.
- Database attributes become more important if you are deploying databases to other platforms. Refer the online help for each database attribute before using it.

## What I Should Know

- How to define a file in the LANSA repository.
- How to add fields to a file definition.
- How to specify file keys.
- How to compile a file to make it operational.
- Some of the attributes that can be specified for database files.

# **REP006** - Logical Views

Logical views are used to create alternate ways of organizing or accessing the data in your LANSA files. You can create file sorting sequences that differ from the keys used in the physical file.

## **Objectives**

- To show how logical views are created over existing files to sequence the file data.
- To add a logical view over the iiiDepartments Department file which sorts the file by iiiDeptDescription.
- To add a logical view to the iiiEmployees file to sort the file by iiiDept Code.

To achieve these objectives, you will complete the following:

Step 1. Add a Logical View to Department File

Step 2. Create Department Test Form

Step 3. Execute Search by Description Form

Step 4. Add a Logical View to Employee File

Summary

# **Before you Begin**

In order to complete this tutorial, you must have completed the previous tutorials.

# Step 1. Add a Logical View to Department File

In this step you will create a very simple logical view over the file iiiDepartments created in the last exercise. This logical view will sort all records in the file by the **iiiDeptDescription** field.

- 1. Use your dynamic list, iiiLIST to open the **iiiDepartments** file in the Editor.
- 2. Select the *Logical Views* tab. It will list all of the defined views for the file (there are none yet).
- 3. Select the *Add* button on the *Home* ribbon and select *Add Logical View* from the list.



A new logical definition is added for the file. You need to specify the logical view details and key fields.

4. The Details tab should already be displayed. Enter the following details:

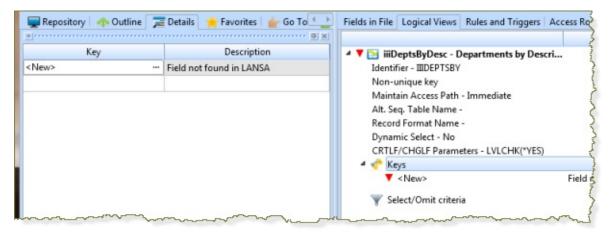
Logical view name **iiiDeptsByDesc** (where **iii**=your initials)

Description **Departments by Description** 

Your *Details* tab should now look like this:

Repository	n Outline 📜 Details 🐈 Favorites 🕍 Go To 👥	Fields in File Logical Views Rules and Triggers Access R
Name Identifier Descriptions	iiiDeptsByDesc IIIDEPTSBY	✓ ▼ ⊡ iiiDeptsByDesc - Depoartments by Desc Identifier - IIIDEPTSBY Non-unique key
English French	Depoartments by Description <new logical=""></new>	Maintain Access Path - Immediate Alt. Seq. Table Name - Record Format Name - Dynamic Select - No
Details Unique Key Access Path	Immediate *	CRTLF/CHGLF Parameters - LVLCHK(*YES)
Dynamic Select		View View

- 5. Click on *<New>* under the *Keys* group so that you can specify the key for the logical view.
  - a. Place the cursor in the Field name field in the Details tab.



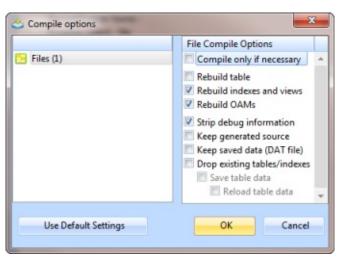
**Note:** A simple way to add key fields is to use the prompt. The prompt will show the *Find* dialog with all valid fields that can be used as key fields shown:

Name	Description	Type	Length	Decimals
iiiDeptCode	Department Code	Alphanumeric	4	
iiiDeptDescription	Department Description	Alphanumeric	20	
			ОК	Cancel

b. Select the **iiiDeptDescription** field, and click OK Your completed logical view will appear like this:

Fields in File	Logical Views	Rules and Triggers	Access Routes	Batch Control	File Attributes	Relationships	Cros
🛛 🛅 iiiDep	tsByDesc - Dep	artments by Descri	pti				
Identit	fier - IIIDEPTSBY						
Non-u	unique key						
Maint	ain Access Path	- Immediate					
Alt. Se	q. Table Name						
Recor	d Format Name	-					:
Dynar	nic Select - No						
CRTLF	F/CHGLF Parame	eters - LVLCHK(*YES)					
4 🦑 Ke	eys						
	iiiDeptDescript	ion	Departmen	nt Description	Ascending (U	Insigned)	
Y Se	lect/Omit criteri	a					
	~			~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~ ~		~

6. Compile the file to make it operational. Remember that you have already compiled the file to build the table and OAM and have added data. Display the *Compile options* dialog. Check that your compile options are set as shown:



#### Note:

- Because you added a Logical View to the file, you have selected to *Rebuild indexes and views* and *Rebuild OAM*.
- In other situations, such as when a new validation rule is added at field level, you will need to select the required *File Compile* options (*Rebuild OAMs* in that particular case).
- If you have changed the fields in the file and need to rebuild the table, the compile is capable of saving and restoring the data to the new file.
- 7. Check that the compile completed successfully.
- 8. Close the file.

# **Step 2. Create Department Test Form**

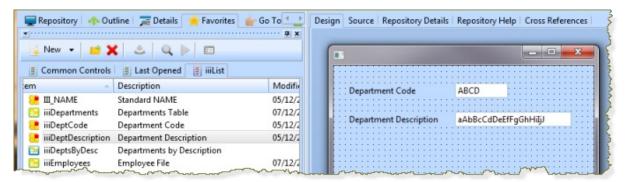
In this step you will create a form that searches the Department file by the department description. You can do this because the logical view you just created is keyed by the department description.

1. Create a form with the following characteristics:

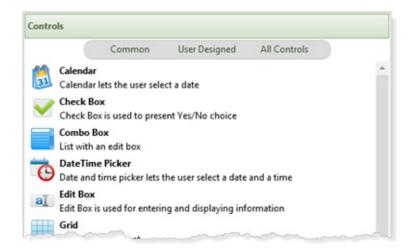
Name	iiiSearchDeptByDesc (where iii=your initials)
Description	Search by Department Description

RDMLX Enabled Select

- 2. Next add the fields iiiDeptCode and the iiiDeptDescription to your form:
  - a. Once the new form has opened in the Editor, display the iiiLIST tab.
  - b. Drag and drop the iiiDeptCode and iiiDeptDescription fields onto the form:



- 3. Next add a button to the form:
  - a. Locate the *Controls* tab. If this tab is not open, open it from the *Home* ribbon, *Views* menu.



b. On the *Controls* tab select the Push button control and drag and drop it onto the form:

Controls				Design	Source	Repository D	Details   R
Dat	Common ta in table format	User Designed	All Controls				
Gro	oup Box oups other controls age ows a picture file				partment (	Code Description	ABCD aAbBc
A Lab				لم.		)	
List	t View I <b>lti-line Edit Box</b> Ilti-line Edit Box has mo	ore than one line of t	ext .				
Par							
Pro Pro	ogress Bar ogress Bar shows how a	n operation progress	es				
Per	sh Button forms an action	)					
(	dio Button dio Button shows mutu	ally exclusive choice	s				

- c. Double-click the button on the form to display the *Details* tab. The *Details* tab is showing the *Properties*, *Events* and *Methods* for the *Push Button*.
- d. In the *Caption* property enter **Find Department Code**.
- e. Adjust the width of the push button to display the Caption.
- f. Adjust the height and width of the form.

Repository 🔥 Outl	ine 📜 Details 🔶 Favorites 🖕	Go To	Design Source	e Repository Details	Repository Help	Cross Referen	nces
*							
PHBN_1		-					
🗉 • 💡							× (
Properties Events Met	hods		Den	rtment Code	ABCD		
Alignment	Center		. Depa	rtment Code	ADCU		1111
ButtonCancel	False						1111
ButtonDefault	False		Depa	rtment Description	aAbBcCdDeEfFg	GhHitjJ	
Caption	Find Department Code						1111
Cursor	*NULL		11111111			::::::::::	::::
P DisplayPosition	3		COL FI	ind Department Code	• : : : : : : : : : : : : :		1111
PragStyle	None				•		
P Ellipses	None						::::

- 4. Next add some code that will search the logical view and return the department code based on the department description:
  - a. Select the *Events* tab on the *Details* tab. Double click on the *Click* event to create a Click event routine for the push button. Your code will look like the following:
  - b. Select the *Source* tab. Your code will look like the following:

EVTROUTINE HANDLING(#PHBN_1.Click) ENDROUTINE.

c. Complete the push button click event routine, by adding a Fetch statement. Your new code should look like the following:

Fetch FIELDS(#iiiDeptcode) from_file(iiiDeptsByDesc) with_key(#iiiDeptDescription)

Your complete source code should now look like this:



- 5. To compile your form:
  - a. Press the *Compile* 👲 button on the Editor toolbar.
  - b. Check that the compile completed successfully.

# **Step 3. Execute Search by Description Form**

In this step you will execute the **Search by Department Description** application. One of the typical uses of logical views is to make a file searchable by a field such as a name.

- 1. Execute form **iiiSearchDeptByDesc** by clicking the *Execute* button on the Editor toolbar.
- 2. Type in one of your department descriptions such as **Administration** and click on the Find Department Code button to retrieve the department code.

Search by Department D	escription 🗖 🗖 📈 🗙
Department Code	ADM
Department Description	Administration
Find Department Code	

The departments descriptions are Administration, Legal, Marketing, Company Training, Business Travel. Remember that the descriptions are case sensitive.

If necessary, execute the iiiMaintDepartment form to see how you entered the descriptions.

**Note:** This is very crude form which simply demonstrates your logical file is working. A logical file of this type (based on a name or description) would usually be used to select a list of departments generically, using a partial key. For example, "show all departments with a description beginning with Ad".

3. Exit the form.

## Step 4. Add a Logical View to Employee File

In this step you will add a logical view to the Employee file to key it by the Department field.

- 1. Open the iiiEmployees file and display the *Logical Views* tab.
- 2. Select the *Designer* button on the ribbon and expand the *Add* menu to select *Add logical view*. Create a logical file:

#### Name: iiiEmpByDeptView

Description: Employee by Department Code.

Details		
Name	iiiEmpByDeptView	
Identifier	ШЕМРВУДЕ	
Descriptions		
English	Employees by Department Code	
French	<new logical=""></new>	
Details		
Unique Key		
Access Path	Immediate	Ŧ
Dynamic Select		
Alt Seq.		
Record Format		
CRTLF/CHGLF Para	ameters	
LVLCHK(*YES)		
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

3. Select the *Keys/New* entry and make the iiiDeptCode field the key.

Field Name	#DentCode	
rielu Name	iiiDeptCode	
Key Order	Ascending	
Numeric Ordering	Unsigned	
Key Position	1	\$
Description	Department Code	
Туре	Alphanumeric	
Length	4	
Decimals		

4. Compile the file and then close it. Use the *Verify / Compile* menu option to ensure that you *Rebuild indexes and views* and *Rebuild OAMs*.

🐣 Compile options	×
Files (1)	File Compile Options Compile only if necessary Rebuild table Rebuild indexes and views Rebuild OAMs Strip debug information Keep generated source Keep saved data (DAT file) Drop existing tables/indexes Save table data Reload table data
Use Default Settin	igs OK Cancel

- 5. Display the iiiLIST and select the iiiEmployees file.
- 6. Right-click and select Properties from the context menu. Notice that the file properties shows brief details of the file such as Date Modified, Task ID and Local Compile State:

Properties of file iiiEmplo	yees	x
Description		
Туре	LANSA file	
Name	iiiEmployees	
Qualifier	D13TRNLIB	
Identifier	IIIEMPLOYE	
Description	Employee File	
Date modified	10/12/2012 16:00:15	
Task ID	*UJI1	
🎂 Compile		
Local Compile State	×	
Repository Enabled for full RDMLX Local Repository State	**	
Master Repository State		Close

You will be using this logical view in a later exercise.

Summary

Important Observations

- Logical views may have one or more key fields.
- Keys can be assigned as ascending or descending sequence.
- Numeric sequence can be signed, unsigned or an absolute value.
- Refer to the online help for more details about these options.

Tips and Techniques

• Logical views provide an efficient means to create sorted indexes to quickly access a file.

What I Should Know

- How to create a logical view.
- How to define the keys to the logical view.
- How to rebuild the file when test data is present.

REP007 - File Validation Rules/Triggers

Application business rules are centrally defined in the LANSA repository as validation rules either in the field or the file definition. Centralized field validations remove the need for developers to code the same validation into each program that uses the same file. A rule is checked when a database file operation (such as an insert, update or delete) is performed using the specified field(s).

Objectives:

- To highlight the difference between field and file level validation rules and the hierarchy which exists between these rules.
- To show how multiple rules can be defined for a field.
- To add the following business rules to file iiiDepartments:

iDeptCode Must be in range A to ZZZ

• To add the following business rules to file iiiEmployees:

iiiDeptCode iiiDeptCode must exist in file iiiDepartments

iiiSalary Must be less than 10000.00

To achieve these objectives, you will complete the following:

- Step 1. Add a Rule to file iiiDepartments
- Step 2. Recompile the File and Test Department Rules
- Step 3. Referential Integrity Rule in iiiEmployees File
- Step 4. Add Rules to iiiSalary Field
- Step 5. Recompile the File and Test Employee Rules
- Step 6. Complete Referential Integrity
- Step 7. Know about File Level Triggers

Summary

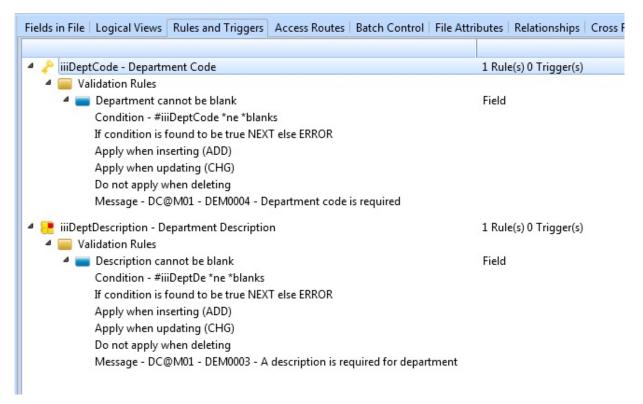
Before you Begin

In order to complete this tutorial, you must have completed the previous tutorials.

Step 1. Add a Rule to file iiiDepartments

In this step you will review the field level validation rule defined on the iiiDeptCode field that you created in exercise REP001 - Create Fields. (When you created the iiiDeptCode field, you copied the validation rules from the DEPTMENT field.) You will also add a rule at file level to demonstrate how these rules interact.

- 1. Open file **iiiDepartments** in the Editor.
- 2. Select the *Rules and triggers* tab.
- 3. Expand the two existing rules to see their complete details. Notice both rules are at field level and cannot be changed using the *Details* tab. (You need to open the field to edit field level rules.)



These rules state that the Department Code and the Department Description codes cannot be blank.

- 4. Select the **iiiDeptCode** field in the list of rules to add a file level rule to it.
- 5. Select *Add* button on the *Home* ribbon, and select an *Add Range Check* rule to the iiiDeptCode field.

		🔶 🖬 -
	Add List Check	
Con	Add Range Check	References
	Add Date Check	vereiences
-	Add Lookup Check	
-	Add Simple Logic Check	er(s)
-	Add Complex Logic Check	
	Add Trigger	
	Add Trigger Condition	

6. In the *Details* tab, create the rule as follows to ensure that the Department Code value has to be between A and ZZZ:

Description	Dept must be A to ZZZ
Sequence	1
When Inserting	Always apply rule (ADD)
When Updating	Apply when field is used (CHGUSE)
When Deleting	Never apply rule
From Value	'A'
To value	'ZZZ'

Your *Details* tab should now look like this:

Details			
Description	Dept m	ust be A to ZZZ	
Sequence	1		\$
Rule Definition			
From Value		To Value	
'A'		'ZZZ'	
V P L P			
Validation Usage	_		
When Inserting	Always	apply rule (ADD)	·*-
When Updating	Apply	when field is used (CHGUSE)	
When Deleting	Never /	Apply Rule	

7. In the *Validation Usage section*, create the rule actions to set the field in error if it is not in the allowed range:

If field is in the range of values	Evaluate next rule (NEXT)
If field is NOT in range of values	Set field in error (ERROR)
Message Number	(delete the entry in this field)
Message File	(delete the entry in this field)
Message Text	Dept must be in range A to ZZZ

Your *Details* tab should now look like this:

Validation Usage		
When Inserting	Always apply rule (ADD)	*
When Updating	Apply when field is used (CHGUSE)	+
When Deleting	Never Apply Rule	+
Actions		
In The Range	Evaluate next rule (NEXT)	+
Not In The Range	Set field in error (ERROR)	*
Error Message		
Message Number		
Message File		
Message Text	Dept must be in range A to ZZZ	

The recommended way to specify the message text is to use a message defined in a Message file as shown in Step 3. Create a Rule for the iiiSalary Field. However, for simplicity in the rest of this tutorial, you will enter the message text in the rule definition.

8. Save the file.

Step 2. Recompile the File and Test Department Rules

Because you have changed a rule (field or file level) in the repository, the OAM of the file must be recompiled.

- 1. Before submitting the compile, check that you do not have any functions or forms active which are accessing the Department file, otherwise you may encounter locking problems.
- 2. Compile the file iiiDepartments. Use the *Verify* menu and select the *Compile* option. Check that your compile options are set as follows:

😃 Compile options	
Files (1)	File Compile Options Compile only if necessary Rebuild table Rebuild indexes and views Rebuild OAMs Strip debug information Keep generated source Keep saved data (DAT file) Drop existing tables/indexes Save table data Reload table data
Use Default Settings	OK Cancel

Note that you have data in the files and you have not changed the table definition (that is, added or removed fields). Also, you have not added or removed any indexes/views. Therefore you only need to rebuild the OAM.

- 3. Check that the file compile completed successfully.
- 4. Execute your Department Maintenance form iiiMaintDepartment.
 - a. Try to add a blank department code. Notice the validation messages which appear and their order (use the up and down buttons in the status bar). You will see both the field and file level messages.

Maintain Departments		
Department Code		
Department Description		
Fetch Insert	Update	Delete
Department code is required		•

- b. Try to add a department code of 111. Notice the error messages.
- c. Exit the application.
- 5. Execute your Employee Maintenance form iiiMaintEmployee.
 - a. Try to add a new employee with a department code of 111. Notice there is no error because the rule was only added to the iiiDeptCode field in file iiiDepartments.

The rule does not apply to the file iiiEmployees.

- b. Close the form.
- 6. Close the file.

Step 3. Referential Integrity Rule in iiiEmployees File

In this step you will add a Lookup rule to the Employee file. This rule specifies that the Department field which is being inserted into the Employee file must exist in the Department file.

- 1. Open file **iiiEmployees** in the Editor.
- 2. On the *Home* ribbon, expand the *Views* menu, to select the *Simple rules list*.



The simple rules list gives you an overview of all the rules in the file and indicates their type and whether a rule is defined in the field or file definition.

Field Name	Level	Seq.	Rule type	Description	Insert	Update	Delete
eiiiDeptCode	Field	1	Simplecheck	Department cannot be blank	Yes	Yes	No
iiiSalary	Field	1	Simplecheck	New simple logic check	Yes	Yes	No
iiiSurname	Field	1	Valuelist	Must not be Blank	Yes	Yes	No

- 3. Select the **iiiDeptCode** field in the list of rules.
- 4. Select the *Designer* button on the ribbon and expand the *Add* menu to select *Add Lookup Check* to add the rule to the iiiDeptCode field.
- 5. In the *Details* tab, create the rule as follows:

Description	Dept must exist in iiiDepartments
Sequence	1
File name	iiiDepartments
When Inserting	Always apply rule (ADD)
When Updating	Apply when field is used (CHGUSE)

Your *Details* tab should now look like this:

Details			
Description	Dept	Dept must exist in Departments	
Sequence	1		÷
Rule Definition			
File Name	iiiDepartments		
iiiEmployees Field or literal		iiiDepartments Key field	
iiiDeptCode		IIIDEPTCO - Department Code	
Validation Usage			
When Inserting	Alwa	ys apply rule (ADD)	
When Updating	_	y when field is used (CHGUSE)	-
			-
When Deleting	Neve	r Apply Rule	

6. Complete the *Actions* section, to define the actions to set the field in error if the value is not found:

If a "key match" is found in target file	Evaluate next rule (NEXT)
If a "key match" is NOT found in target file	Set field in error (ERROR)
Message file	(delete the entry in this field)
Message Number	(delete the entry in this field)
Message text	Department must exist in Department File

Your *Details* tab should now look like this:

Actions	
Key Match Is Found	Evaluate next rule (NEXT)
Key Match Is Not Found	Set field in error (ERROR)
Error Message	
Message Number	
Message File	
Message Text	Department must exist in Departments Table

7. Save the file.

Step 4. Add Rules to iiiSalary Field

In this step you will create a validation check on the iiiSalary field to ensure that its value is less than 10000. (The field already has a rule stating that it cannot be 0). The rule will be a range check.

- 1. The **iiiEmployees** file should be open in the Editor. The *Rules and trigger* tab and *Details* tab should be visible.
- 2. Select the **iiiSalary** field in the list of rules.
- 3. Select the *Designer* button on the ribbon and expand the *Add* menu to select *Add Range Check*.

Description	Amount must be less than 10000
Sequence	1
When Inserting	Always apply rule (ADD)
When Updating	Apply when field is used (CHGUSE)
When Feleting	Never apply rule
From Value	0.01
To Value	10000

4. In the *Details* tab, create the rule as follows:

5. Complete the *Actions* section, to define the list check rule as follows:

If field is in the range of values	Evaluate next rule (NEXT)
If field is NOT in range of values	Set field in error (ERROR)
Message File	(delete the entry in this field)
Message Number	(delete the entry in this field)
Message Text	Amount must be less than 10000

6. Save your file definition.

Step 5. Recompile the File and Test Employee Rules

In this step you will recompile the Employee file so that the new rules will be applied.

- 1. Before submitting the compile, check that you do not have any forms active which are accessing the Employee file, otherwise you may encounter locking problems.
- 2. View the rules for the Employee file. Notice it now has both Field and File Level rules.

Field Name	Level	Seq.	Rule type	Description	Insert	Update	Delete
iiiDeptCode	Field	1	Simplecheck	Department cannot be blank	Yes	Yes	No
iiiDeptCode	File	1	Lookup	Dept must exist in Departments	Yes	When used	No
iiiSalary	Field	1	Simplecheck	New simple logic check	Yes	Yes	No
iiiSalary	File	1	Rangecheck	Amount must be less than 10000	Yes	When used	No
iiiSurname	Field	1	Valuelist	Must not be Blank	Yes	Yes	No

3. Recompile the **iiiEmployees** Employee file.

Note: Once again that the changes made to the Employee file concern file level validation rules. Therefore you need to recompile the OAM. The last compile action used this option. If you click on the *Compile* button, the same option will be used again.

Remember to check the correct compile options are used by using the *Verify / Compile* **option.**

- 4. Check that the file compile completed successfully.
- 5. Execute your Employee Maintenance form iiiMaintEmployee.
 - a. Fetch an employee and try to update the details with a Department (such as XXX) that does not exist.

Employee Number	01
Surname	Brown
Given Name	Veronica
Department Code	XXXX
Salary	00000500000
Start Date	01/12/2012 *
Employee Notes	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean quam lorem, aliquam in egestas a, luctus id elit. Nulla quis justo leo, a convallis dui. In ac dui ante, venenatis aliquet purus. Nulla mattis porta accumsan. Vivamus enim ipsum, ullamcorper at aliquam ut, consectetur a diam. Suspendisse ornare mattis massa vitae
Fetch	Insert Update Delete

b. Fetch an existing record and try to change the Salary to 10001.

If these entries weren't rejected, check the rules that you created during the previous steps.

- 6. Exit the application.
- 7. Close the file in the Editor.

Step 6. Complete Referential Integrity

In this step you will complete the referential integrity checks. You will add a rule that checks if any employees exist for a specific department before allowing the department to be deleted from the Department File. This rule will prevent an employee from losing its parent department.

- 1. Open file **iiiDepartments** in the Editor.
- 2. Select the *Rules and Triggers* tab.
- 3. Select the key field **iiiDeptCode** field in the list of rules.
- 4. Select the *Designer* button on the ribbon, and expand the *Add* menu to select *Add Lookup Check* to add the rule to the iiiDeptCode field.
- 5. In the *Details* tab, create the rule as follows:

Description	Cannot delete if Employee exists
Sequence	2
File Name	iiiEmpByDeptView
When Inserting	Never apply rule
When Updating	Never apply rule
When Deleting	Always apply rule (DLT)

Use the ellipsis button for File Name to use the Find dialog :

66 Files	· · · · · · · · · · · · · · · · · · ·	×
Like name	IIIEMP	
Like description		
		Find Find
Name	Description	Library
🕑 🛅 iiiEmpByDeptV	ew Employees by Department Code	D13TRNLIB
iiiEmployees	Employee File	D13TRNLIB
		OK Cancel

Note that the rule is only enforced when deleting a record. Your *Details* tab should now look like this:

Description	Cannot delet if Emp exists	
Sequence	2	
Rule Definition		
File Name iiiEmpByDeptView		
iiiDepartments Field or literal	iiiEmpByDeptView Key field	
iiiDeptCode	IIIDEPTCO - Department Code	
Validation Usage		
	Never Apply Rule	Ţ
When Inserting	Never Apply Rule Never Apply Rule	•
When Inserting When Updating		•
Validation Usage When Inserting When Updating When Deleting Actions	Never Apply Rule	•
When Inserting When Updating When Deleting	Never Apply Rule	*

6. In the *Actions* section, define the File Lookup Rule to set the field in error if the department exists in the Employee file:

If a "key match" is found in target file	Set field in error (ERROR)

If a "key match" is NOT found in target file	Evaluate next rule (NEXT)
Message file	(delete the entry in this field)
Message Number	(delete the entry in this field)
Message text	Cannot delete if used in Employee file

Your *Details* tab should now look like this:

Key Match Is Found	Set field in error (ERROR)	
Key Match Is Not Found	Evaluate next rule (NEXT)	
Error Message		
Message Number		
Message File		
Message Text	Cannot delete if used in Employ	yee file

7. Recompile the file **iiiDepartments**.

Check the file compile completed successfully.

- 8. Execute your Department Maintenance form iiiMaintDepartment.
 - a. Try to delete department **ADM**. You will not be able to delete this record.

Maintain Departments	
Department Code	ADM
Department Description	
Eetch	nsert Update Delete
Cannot delete if used in Employ	ee file

b. Add a new department XYZ. Delete this department. The delete will be

allowed because no employees have been defined for this department. 9. Exit the application.

10. Close the file.

Step 7. Know about File Level Triggers

A database trigger is a condition that is defined for a field or file in the repository. Typically triggers are used to move complex logic to the file definition. The trigger definition links a trigger function to a specific database operation and condition. This function is invoked **automatically** when a specific type of I/O operation occurs to a file and when a specific set of conditions are met.

Similar to validation checks, LANSA triggers centralize the business logic of your application.

A LANSA trigger function is a special type of LANSA function, which is invoked automatically when a specific type of I/O operation occurs on a file (Open, Close, Read, Insert, Update and Delete) and when a specific set of conditions are met.

For example, as an audit trail, you might create a trigger to print before and after recording specific values.

Triggers are not covered in this tutorial because you need to write code in order to create a trigger function.

Summary

Important Observations

- Field level validation rules are applied before file level validation rules.
- The sequence of rules, combined with the validation actions, is very important in controlling when multiple rules are applied.
- Referential file integrity can be added by using file lookup checks. Remember that referential integrity is usually defined in both files within a relationship. In our example, Department cannot be deleted if Employees exist, and Employees cannot be added unless the Department exists.
- Notice that the trigger function is called after the validation rules are performed. This order of operations prevents invalid data from being passed into the trigger.
- Whenever a validation rule or trigger is changed (at field or file level), the file OAM needs to be recompiled.

Tips and Techniques

- Most business rules should be defined at the file level. Only in special circumstances should you use a field level rule. For example, when you define the rule for iiiSalary at the field level, then every file using the iiiSalary field has the same rules. Quite often, the iiiSalary will have a different meaning and different rules when used in a different file.
- You can create Complex logic rules when you need to perform more complex checking than supported by standard value types.
- The use of NEXT, ERROR and ACCEPT processing with multiple rules for a field is very important. ACCEPT processing means that no more rules are evaluated.
- It is very important that you check the *Actions* tab to ensure that you have properly defined the validation usage, that is, when should an error occur. For example, a simple logic check rule can be defined as (#DEPT *NE *BLANKS). If true, is this an error?
- The ADDUSE and CHGUSE options control if validation rules are performed. If the field is not included in insert, update or delete, the rule check will not be performed. Typically CHGUSE is applied on file rules (not ADDUSE or DLTUSE). For some additional explanation of ADDUSE and CHGUSE, refer to the *Important Observations* in the Summary of the

REP008 - *Virtual Fields* exercise. This exercise gives an example of how these options can be used.

- Trigger functions are very useful if you want to perform server-side procedures when building client/server or web-based applications.
- Trigger functions can be used to define virtual fields and can be used to perform complex validation rules in a file.

What I Should Know

- Where validation rules are performed in the LANSA architecture.
- What rules are supported at the file level.
- At what level you should specify most of your rules.
- In what order are the field and file validation rules performed.
- How to use a code file/table lookup rule to create referential integrity.
- That you need to recompile your file after you have changed it.

REP008 - Virtual Fields

A virtual field is a field which appears as part of a file but does not actually exist in the physical database file. Its value is dynamically derived based on the fields in the file. Virtual fields provide flexibility, since you can combine or extract data from fields in a file and extend the information already contained in the file.

Objectives:

- To explain virtual fields.
- To highlight the different types of virtual fields that can be defined using the LANSA Repository.

To achieve these objectives, you will complete the following:

Step 1. Add Virtual Fields to Employee File

Step 2. Calculation Virtual Field

Step 3. Concatenation Virtual Field

Step 4. Code Fragment Virtual Fields

Step 5. Create Test Form

Summary

Before you Begin

In order to complete this tutorial, you must have completed the previous tutorials.

Step 1. Add Virtual Fields to Employee File

In this exercise, you will see how the multi-add field feature can be used to create fields in the repository as they are added to the file. You will use these new fields as virtual fields in the Employee file.

- 1. Open file **iiiEmployees** in the Editor.
- 2. In the Fields in File tab, Select the *Designer* button on the ribbon and expand the *Add* menu to select *Multi-add fields*. The *Add Fields to the File* window is displayed.
- 3. Add the following fields **(where iii=your initials)**. Remember to tick the Virtual check box.

Field Name	Description	Туре	Length	Dec	Virtual
iiiTaxAmount	Income Tax to Deduct	Packed	6	2	ü yes
iiiFullEmployNumber	Full Employee Number	Alpha	9		ü yes
iiiEmployYears	Years with Company	Packed	2		ü yes
iiiNoteLength	Length of Notes as Characters	Packed	5		ü yes

The autocomplete feature will attempt to match existing fields in the repository and will leave default values that need to be changed. Be sure to enter the correct descriptions, type, length, decimals and set the virtual flag.

Field Name	Description	Reference Field	Туре	Length	Decs	Key	Virt
iiiTaxAmount	Income Tax to Deduct		Packed	6	2		•
iiiFullEmployNumber	Full Employee Number		Alpha	5			☑
iiiEmployYears	Years with the Company		Packed	2			☑
iiiNoteLength	Length of Notes as Charac		Packed	5			
			-	ОК		Cance	

4. Press OK.

Field Not Foun	d in LANSA
Name	iiiTaxAmount
Description	Income Tax to Deduct
Туре	Packed
Length	6
Decimals	2
Identifier	ШТАХАМО
Create	Create All Ignore Cancel

- 5. When prompted to create the fields, click the *Create All* button.
- 6. Your new fields in the file will appear like this:

🔻 🚼 iiiTaxAmount	Income Tax to Deduct	Packed	6	2
🔻 🚼 iiiFullEmployNumb	er Full Employee Number	Alpha	5	
🔻 🚼 iiiEmployYears	Years with the Company	Packed	2	
▼ 🔡 iiiNoteLength	Length of Notes as Characters	Packed	5	

- 7. You need to make some modifications to your new fields in the Repository:
 - a. Right-click **iiiTaxAmount** field and choose the *Open* option from the context menu:

Undefined Virtuals V iiiTaxAmount	Income Tax to Deduct			Pac
▼ 🔐 iiiFullEmp 🕂 ▼ 🔐 iiiEmploy 🖃 ▼ 🔐 iiiNoteLer	Expand all			Alpi Pac Pac
	iiiTaxAmount - Income Tax to Deduct	13	Open	
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~_~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	×	Delete from Repository Quick Export Check In	

- b. Make the *Edit Mask* **A**.
- 8. Save and close the field.
- 9. Open the **iiiNoteLength** field in the Editor.
  - a. Change the *Edit Mask* to be 2.
- 10. Close and save the field.

Note that there are warning messages for the new virtual fields indicating they do not have a derivation logic.

## **Step 2. Calculation Virtual Field**

In this step you will define the iiiTaxAmount virtual field to be calculated as 25% of the iiiSalary field:

- 1. Select the **iiiTaxAmount** field in the list of *Undefined virtuals*.
- 2. In the *Details* tab, enter the following:

Virtual field type	Calculation
Derive value when record is read	ü Select option for yes

Factor	Operation	Factor	Result
iiiSalary	/	100	*WORKFLD
*WORKFLD	*	25	iiiTaxAmount

Your *Details* tab should now look like this:

Field Name	iiiTaxAmount		
Sequence	1		÷
/irtual Field Type	Calculation		-
Derive When Derive value w	hen record is re	ad	
Derive value w	hen record is re	ad	
Derive value w	hen record is re Op.	ad Factor	Result
Derive value w Derivation			Result *WORKFLD

Note:

• Calculation Virtual Fields can only be calculated when a record is read.

- They are Display-only Virtual fields.
- 3. Now that its derivation has been defined, the iiiTaxAmount field will be listed under *Read virtuals*.

Rea	d Virtuals				
8	iiiTaxAmount	Income Tax to Deduct	Packed	6	2

## Step 3. Concatenation Virtual Field

In this step you will concatenate the Department field iiiDeptCode with the Employee Number field iiiEmployNumber to set the value of the Full Employee Number field iiiFullEmployNumber.

- 1. Select the **iiiFullEmployNumber** field in the list of *Undefined virtuals*.
- 2. In the *Details* tab, enter the following:

Virtual field type	Concatenation
Derive value when record is read	Select box for yes
Populate real field when writing to the fil	e No
Field name	iiiDeptCode iiiEmployNumber

Your *Details* tab should now look like this:

Field Name	iiiFullE	iiiFullEmployNumber				
Sequence	2	2				
Virtual Field Type	Conca	Concatenation				
Derive When						
Derive value wh	en reco	rd is read				
Populate real field when writing to the file						
Derivation		_				
Field Name		Length	Description			
iiiDeptCode		4	Department Code			
iiiEmployNumber …		5	Employee Number			

3. The iiiFullEmployNumber field will now be listed under *Read virtuals*.

## **Step 4. Code Fragment Virtual Fields**

In this step you will use a Code Fragment derivation to:

- Assign the length of the Notes field to the iiiNoteLength virtual field and
- To set the value of the iiiEmployYears fields to be the number of years the employee has been with the company.
- 1. Select the **iiiNoteLength** field in the list of *Undefined virtuals*.
- 2. In the *Details* tab, enter the following:

Virtual field type	Code Fragment
Derive value when record is read	Select for Yes
Virtual Field Derivation	#iiiNoteLength := #iiiEmployNotes.CurChars
	Replace iii with your initials.

Your *Details* tab should now look like this:

Details 🖸				
IIIFILEMP				
Field name	IIINOTLEN			
Description	Note Lenght			
Sequence	3			
Virtual field type	Code fragment			
<ul> <li>Derive value when record is read</li> <li>Populate real field when writing to the file</li> </ul>				
Virtual Field Derivation				
00001 00002	#IIINOTLEN:= #iiinotes.CurChars			

Details				
Field Name	iiiNoteLength	•]		
Sequence	3			
Virtual Field Type	Code fragment	-		
Derive When				
Derive value when record is read				
Populate real field when writing to the file				
Derivation				
Virtual Field Deriv	vation			
#iiiN	oteLength := #iiiEmployNotes.curchars	]		
hanne	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	JL		

The code fragment uses an RDMLX intrinsic method CurChars to determine the number of characters in the iiiEmployNotes field and assigns the result to the iiiNoteLength field.

3. The iiiNoteLength field will now be listed under the *Read virtuals*.

Next you will set the iiiEmployYears field to show how many years an employee has worked for the company based on the Start Date field.

- 4. Select the **iiiEmployYears** field in the list of *Undefined virtuals*.
- 5. In the *Details* tab, enter the following:

Virtual field type	Code Fragment
Derive value when record is read	ü Select this option
Virtual Field Derivation	<pre>#iiiEmployYears := #iiiStartDate.Date.Now.Difference(#iiiStartDate) / 365</pre>
	(you can copy and paste this code from the <i>online guide Tutorials</i> , substituting your initials for iii)

Your *Details* tab should now look like this:

Details					
Field Name	iiiEmployYears …				
Sequence	\$				
Virtual Field Type	Code fragment *				
Derive When					
Derive value when record is read					
Populate real field when writing to the file					
Derivation					
Virtual Field Deriv	vation				
#iiiE	<pre>mployYears := #iiistartdate.date.now.difference(#iiiStartdate.date) / 365</pre>				

The code fragment uses RDMLX intrinsic methods Date and Difference to compare the date portion of the iiiStartDate field with today's date and divides the result by 365.

All your virtual fields are now listed under Read Virtuals.

JES	Before				
R	ead Virtuals				
> [	iiiTaxAmount	Income Tax to Deduct	Packed	6	2

6. Compile the file and ensure the compilation ends without errors.

#### **Step 5. Create Test Form**

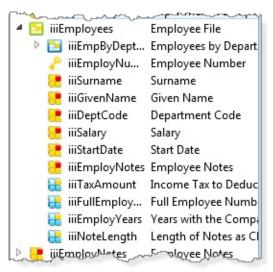
In this step you will create a form that will show the real fields and the virtual fields in the Employee file.

- 1. Create a form.
  - a. Enter the following characteristics for the form:

Name	iiiMaintEmployVirtuals	(where iii=your initials)
Description	Real and Virtual Fields in	Employee File

RDMLX Enabled **ü Yes** 

- 2. Once the *Editor* has opened, make your form wider.
- 3. Display the iiiLIST tab and expand the iiiEmployees file to display the fields it contains. Notice that the real fields and the virtual fields have different icons:



- 4. Drag and drop all the real fields to your form. Select them as a group using the *Shift* + *Left Mouse button*.
- 5. Drag all the virtual fields next to the real fields.

Employee Number	ABCDE	11	ncome Tax to Deduct	1,234.12
Surname Given Name	aAbBcCdDeEfFgGhHiljJkKILm aAbBcCdDeEfFgGhHiljJkKILm		Full Employee Number Years with the Company	ABCDEFGHI : : : : : : : : : : : : : : : : : : :
Department Code	ABCD		ength of Notes as Characte	ers 12,345
Start Date	01/01/1900 💌 aAbBcCdDeEfFgGhHiljJkKlLmMnNoOpPqQ 🔺			
Employee Notes	rRsStTuUVVWWXXYZZaAbBcCdDeEfFgGhH iIjJkKILmMnNoOpPqQrRsStTuUVWWXXYZ ZaAbBcCdDeEfFgGhHIJJkKILmMnNoOpPq QrRsStTuUVWWXXYZZaAbBcCdDeEfFgGh HiIjJkKILmMnNoOpPqRrsStTuUvVWXXY YzZaAbBcCdDeEfFqGhHiIjJkKILmMnNoOp			
•	YZZƏADBCCdDetrqunHilykkilmininoOp			

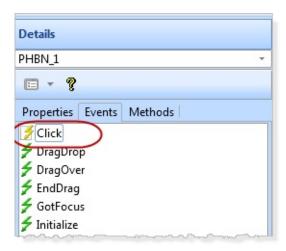
6. Display the *Controls* tab and drag and drop a push button to the form.

Contro	ols					
	Common	User Designed	All Controls			
A	Lapei Text description	ı		*		
:::	List View List View					
aaa aa i	Multi-line Edit Box Multi-line Edit Box has more than one line of text					
	Panel Container for other components					
	Progress Bar Progress Bar shows how an operation progresses					
	Push Button Performs an ac	tion	)			
۲	Radio Button Radio Button sł	nows mutually exc	clusive choices			
1 mil	Shortcut Key Shortcut Key		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~			

7. With the push button select, on the *Details* tab, make the *Caption* of the button **Fetch**.

Details			
PHBN_1			-
E • ?			
Properties	Events	Methods	
P Alignment		Center	
P Buttor	nCancel	False	
Buttor	Default	Faise	
Captio	n	Fetch	- )
Cursor		*NULL	
🖻 DisplayPositio		12	=
DragStyle		None	
C Ellipses		None	

8. Display the Events tab and double click on the Click event to create an Click event routine for the button.



9. Display the Source tab of your form and add this code inside the event routine for #PHBN_1.Click replacing III with your initials:

EVTROUTINE HANDLING(#PHBN_1.Click)

Fetch FIELDS(*all) from_file(iiiEmployees) with_key(#iiiEmploynumber) ENDROUTINE

Your form is now ready to compile.

10. Compile and execute your form.

Real and Virtual Fields i	in Employee File	
Employee Number Surname Given Name Department Code Salary Start Date	01 Brown Veronica ADM 5000.00 01/12/2010	Income Tax to Deduct1,250.00Full Employee NumberADM 01Years with the Company02Length of Notes as Characters381
Employee Notes	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean quam lorem, aliquam in egestas a, luctus id elit. Nulla quis justo leo, a convallis dui. In ac dui ante, venenatis aliquet purus. Nulla mattis porta accumsan. Vivamus enim ipsum, ullamcorper at aliquam ut, consectetur a diam. Suspendisse ornare mattis massa vitae adipiscing. Sed sed neque enim,	
Fetch		

- a. Enter an Employee number and click *Fetch*.
- b. Note how the virtual field values are based on the real field values.
- 11. Exit the form.
- 12. Close the form in the *Editor*.

## Summary

#### **Important Observations**

- Virtual fields must be first be defined in the Repository. The virtual fields can then be added to the file definition.
- Calculation virtuals can only be defined as *Derive value when record is read*. The other virtuals can use be defined as *Derive value when record is read* and *Populate real field when writing to the file*.
- Virtual fields can have validation rules just like real fields.

# **Tips and Techniques**

- The Multi-add fields feature can add both real and virtual fields.
- Virtual fields cannot be used as key fields.
- Triggers provide another option for defining virtual fields in a file when more complex coding is required.

# What I Should Know

- What are the different types of virtual fields.
- How to add virtual fields to a file.

# **REP009** - Access Routes and Predetermined Join Fields

LANSA uses **access routes** to describe relationships between files in a database. Access routes are simply descriptions which answer questions such as "How do I access the associated records in file B, given that I have a record from file A?" They have no physical impact on the database.

A **predetermined join field** (PJF) is a special kind of virtual field based on access routes whose value is determined by a field in another file. (Ordinary virtual fields are always based on fields within its own file definition.)

# **Objectives:**

- To create an access route to define the relationships between the Department and Employee files
- To show how to create and maintain predetermined join fields (PJFs).
- To add a calculated predetermined join field to the Department File based on the fields in the Employee file.
- To add a file lookup Predetermined Join Field to the Employee File based on the fields in the Department file.

To achieve these objectives, you will complete the following:

Step 1. Understand the Database Relationship

Step 2. Create Access Route from Department File

Step 3. Create Access Route from Employee File

Step 4. Create Fields in the LANSA Repository

Step 5. Add File Lookup PJF to Employee File

Step 6. Test New PJF in Employee File

Step 7. Add PJFs to Department File

Step 8. Modify Test Form

Summary

## **Before you Begin**

In order to complete this tutorial, you must have completed the previous tutorials.

## Step 1. Understand the Database Relationship

Before creating an access route, you need to understand the relationship between the Department (iiiDepartments) and the Employee (iiiEmployees) files that you have already created. The company structure is a hierarchy of departments and employees:

Departments	ADM	LGL	TRN	TRV
-------------	-----	-----	-----	-----

Employees	01,02	03	04	
-----------	-------	----	----	--

Departments have employees. An employee cannot exist without the department. Stated another way, the department is the parent or owner of its employees.

Relationships work in both directions. From the perspective of the Employee file, it is the child of the Department. An employee belongs to just one Department.

## Step 2. Create Access Route from Department File

In this step you will create an access route from the Department file to the Employee file to describe the file relationship.

- 1. Open file **iiiDepartments** in the Editor.
- 2. Select the *Access Routes* tab.
- 3. Click the *Add* button on the *Home* ribbon and select *Add Access Route* from the menu.

**Note:** The ribbon is context sensitive. *Add Access Route* is the only valid option at this point.

4. In the *Details* tab, enter the following characteristics:

Access route name	e iiiTOEMP (where iii=your initials)
Description	To Employee File
Accessed file	iiiEmpByDeptView ***
Maximum records	More than one
Keep last	
Default action	Ignore and continue processing (IGNORE)
Derivation:	After virtual fields

*** Use the ellipsis button and *Find* dialog to select the logical file.

Your *Details* tab will look like this:

Details		
Access Route Name	ШТОЕМР	
Description	To Employees File	
Accessed File	iiiEmpByDeptView	
Association Type	Derive from Maximum Records and Default Action	*
Association Rule	Derive From Default Action	+
Documentation Only		
Maximum Records	More than one	*
Default Action	Ignore and continue processing (IGNORE)	*
Keep Last		¢
Derivation	After virtual fields	*

5. Use the *Add* button on the *Home* ribbon, to select *Add Key*. Use the ellipsis button and then the *Find dialog* to select the key field from the Department File, iiiDeptCode

Key Field/Value	iiiDeptCode	
Description	Department Code	
Гуре	Alphanumeric	
Length	3	
Decimals		
Key Position	1 \$	
Target Key		
Name	IIDEPTCO	
Description	Department Code	
Туре	Alphanumeric	
Length	3	
Decimals		

**Note:** The Target Key has been recognized and selected in the Department file.

6. Your *Access Route* definition should now look like the following.

Fields in File   Logical Views	Rules and Triggers	Access Routes	Batch Control	File Attributes	Relati
					į
🛚 💥 IIITOEMP - To Emplo	yees File				1.11
File accessed - iiiEmpE	ByDeptView - Employ	ees by Departme	ent Code		1
Association Type - De	rive from Maximum	Records and Defa	ault Action		
Association Rule - Der	ive From Default Act	tion			
Access Route is Action	nable				
More than 1					
Default action - Ignor	and continue proce	essing (IGNORE)			
Keep last 0 records					
4 🦑 Keys					
📒 IIIDEPTCODE				Department Co	de
🔒 PJFs derived after	virtual fields				
	mm	,	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	

7. Save and close the file.

## Step 3. Create Access Route from Employee File

In this step you will create an access route from the Employee file to the Department file to describe the file relationship.

- 1. Open file **iiiEmployees** in the Editor.
- 2. Select the Access Routes tab. Ensure that the Details tab is visible.
- 3. Use the *Add* button on the *Home* ribbon, to select *Add Access Route* from the menu.
- 4. In the *Details* tab, enter the following characteristics:

Access route name iiiTODEPT

Description	To Department File
Accessed file	iiiDepartments
Maximum records	One
Keep last	
Default action	Abort and issue an error message (ABORT)
Derivation:	After virtual fields

- 5. Click on *Keys / New* and use the ellipsis button and *Find dialog* to select the *Key Field/Value* to lookup a Department as,**iiiDeptCode**.
- 6. Your *Access Route* definition should look like the following:.

Details		Fields in File   Logical Views   Rules and Triggers   Access Routes   Batch Control   File Attributes   Relationship
Key Field/Value Description Type	iiiDeptCode Department Code Alphanumeric	IIITODEPT - To department File File accessed - iiiDepartments - Department Table Association Type - Derive from Maximum Records and Default Action Association Rule - Derive from Default Action
Length Decimals Key Position Target Key	4  1	Access Route is Actionable Number of records - 1 Default action - Ignore and continue processing (IGNORE) Keep last 0 records
Name Description	IIIDEPTCO Department Code	<ul> <li>✓ ⁴ Keys</li> <li>● iiiDeptCode</li> <li>Department Code</li> <li>B PJFs derived after virtual fields</li> </ul>
Type Length	Alphanumeric 4	
Decimals		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

7. Save and close the file.

# Step 4. Create Fields in the LANSA Repository

In order to add a *Predetermined Join Field* to a file definition, the field has to first be defined in the *Repository*. In this step you will create two field definitions which you will use when creating your *Predetermined Join Fields*.

- 1. Create two fields called iiiTotalSalaryPJF and iiiDeptDescPJF. The iiiTotalSalaryPJF field will contain the total value based on the iiiSalary field. The iiiDeptDescPJF field will hold the Department Description based on the iiiDeptDescription field.
- 2. In the most efficient means possible, create the following fields with the basic characteristics listed below. (Reminder: **iii**=your initials.)

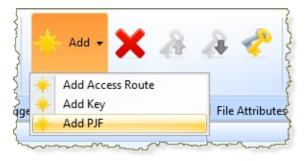
Field name:	iiiDeptDescPJF
Description:	Department Description
Туре:	Alpha
Length:	20

Field name:	iiiTotalSalaryPJF
Description:	<b>Total Salaries</b>
Туре:	Packed
Length:	11
Decimals:	2
Edit mask:	Α

# Step 5. Add File Lookup PJF to Employee File

In this step you will add a PJF for the Department Description field from the iiiDepartments Department file to the iiiEmployees Employee file. By defining a PJF (Predetermined Join Field) in the file, you can access the Department Description each time the Employee file is accessed without writing extra code to access the Department file. This field will appear as if it is simply part of the Employee file, and will be retrieved by the OAM.

- 1. Open file **iiiEmployees** in the Editor.
- 2. Ensure that the *Fields in File* tab is visible. Notice that no PJFs have been defined.
- 3. Select the *Access Routes* tab to add your new predetermined join fields.
- 4. Select the **IIITODEPT** access route. Use the *Add* button on the *Home* ribbon, and select *Add PJF* from the menu.



- 5. Complete the *Details* tab, by enter the following characteristics:
  - PJF field iiiDeptDescPJF
  - PJF Type Lookup
  - Source field iiiDeptDescription

Use the ellipsis button and *Find dialog* to select each field.

Your *Details* tab will now look like this:

🛧 Outline 📻	Details 🍵 Favorites 🖕 Go To 🛛 🔯 Featur	
PJF Field		×
PJF Type	Lookup	
PJF Field	iiiDeptDescPJF	
Description	Department Description	
Туре	Alphanumeric	
Length	20	
Decimal Places		
Target Field		
Source Field	iiiDeptDescription	•••
Description	Department Description	
Туре	Alphanumeric	
Length	20	
Decimal Places		

The value of the iiiDeptDescription from the Department file will now be retrieved and returned to a LANSA program as the field iiiDeptDescPJF in the Employee file.

**Note:** Predetermined Join Fields are display only fields.

Your access route should now appear like this:

ields in File   Logical Views   Rules and Triggers	Access Routes	Batch Control	File Attributes	Relationships	Cross References	Virtual Derivation
🛚 💥 IIITODEPT - To Department File						
File accessed - iiiDepartments - Departmen	t Table					
Association Type - Derive from Maximum F	Records and Defa	sult Action				
Association Rule - Derive From Default Acti	ion					
Access Route is Actionable						
Number of records - 1						
Default action - Abort and issue an error m	essage (ABORT)					
Keep last 0 records						
4 🦑 Keys						
IIIDEPTCODE			Department Co	de		
PJFs derived after virtual fields						
🔒 iiiDeptDescPJF			Department De	scription		LOOKUP fie

6. Select the *Fields in File* tab. Your field list should now appear like this:

field Name	Description	Ref. Field	Туре	Length	Decimals
Primary Keys					
🔑 iiiEmployNumber	Employee ID		Alpha	2	
Real Fields					
iiiEmployNumber	Employee ID		Alpha	2	
📒 iiiSurname	Surname	III_NAME	Alpha	25	
: iiiGivenName	Given Name	III_NAME	Alpha	25	
:iiDeptCode	Department Code		Alpha	3	
: iiiSalary	Monthly Salary		Packed	11	2
🔻 📑 iiiStartDate	Start Date		DateTime	26	
📑 iiiEmployNotes	Employee Notes		String	512	
PJFs Before					
Read Virtuals					
iiiTaxAmount	Income Tax to Deduct		Packed	6	2
E iiiFullEmployNumber	Full Employee Number		Alpha	9	-
iiiEmployYears	Years with the Company		Packed	2	
iiiNoteLength	Length of Notes as Characters		Packed	5	
PJFs After	Longin of Hotes of Childrenes		. senea	-	
iiiDeptDescPJF	Department Description		Alpha	20	
Write Virtuals					
Inactive Virtuals					
Undefined Virtuals					

7. Select the Select View Style button and change the display to *Fields List View* to get a different overview of your file:



Your *Fields in File* tab will now look like this:

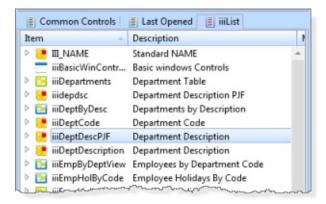
Fields in File Log	ical View	s Rules	and Triggers   Access R	outes   Batch Co	ontrol	File Attributes	Relation	ships	Cross Re	ference	s V «	100
Field Name	Seq. 🔺	Key	Description			Ref. field	Тур	e	Lengt	h Decs		_
iiiEmployNu		1	Employee ID				Alp		2			
iiiSurname	2		Surname			III NAME	Alp		25			
iiiGivenName	3		Given Name			III NAME	Alp		25			
iiiDeptCode	4		Department Code				Alp		3			
iiiSalary	5		Monthly Salary					ked	11	2		
iiiStartDate	6		Start Date				Dat	eTime	26			
iiiEmployNo	7		Employee Notes				Stri	ng	512			
irtual Fields						1.						
Field Name	Se	Descrip		Ref.	field	Туре	Length		Derive	When		
Field Name 膏 iiiTaxAmount	1	Incom	Tax to Deduct	Ref.	field	Packed	6	Decs 2	Read	When		
Field Name EiiiTaxAmount EiiiFullEmploy	1 2	Incom Full Em	e Tax to Deduct ployee Number	Ref.	field	Packed Alpha	6 9		Read Read	When		
Field Name 膏 iiiTaxAmount	1 2 3	Incom Full Em Years v	Tax to Deduct	Ref.	field	Packed	6		Read	When		
Field Name iiiTaxAmount iiiFullEmploy iiiEmployYears iiiNoteLength redetermined Joir	1 2 3 4	Income Full Em Years v Length	tax to Deduct ployee Number vith the Company	Ref.		Packed Alpha Packed Packed	6 9 2 5	2	Read Read Read Read			
Field Name iiiTaxAmount iiiFullEmploy iiiEmployYears iiiNoteLength	1 2 3 4 Fields	Income Full En Years v Length	tax to Deduct ployee Number vith the Company	Ref.		Packed Alpha Packed	6 9 2	2	Read Read Read Read	When	Source	

- 8. Click the iiiDeptDescPJF field. Notice that it is possible to edit the PJF definition from the *Fields in File* tab.
- 9. Compile the file and close it.

# Step 6. Test New PJF in Employee File

In this step you will add the predetermined join field Department description to iiiMaintEmployVirtuals (Real and Virtual Fields in Employee File).

- 1. Using the iiiLIST tab, locate and open the form iiiMaintEmployVirtuals.
  - a. Display the *Design* tab.
  - b. In the *iiiLIST* tab locate the **iiiDeptDescPJF** (Department Description) field.



c. Drag the iiiDEPPJF field onto the form above the virtual fields. You may need to move the existing virtual fields lower.

**Hint:** Select the four virtual fields using the Shft+Left mouse button, and drag them lower with the Ctrl+Cursor key.

Employee ID	AB		Department	Description	ABCDEFGHUK	LMNOPQ	RST
Sumame	aAbBcCdDeEf	FgGhHiljJkKILm		Income Tax t	o Deduct	1.234.12	
Given Name	aAbBcCdDeEf	FgGhHiljJkKILm		Full Employe	e Number	ABCDEF	GHI
Department Code	ABC			Years with th		12	
Monthly Salary	123,456,789.12				tes as Characte	Hrs 12	345
Start Date	01/01/1900					::::::	
	rRsStTuUvVwV	gGhHiljJkKlLmMnt hXyYzZaAbBcCdDe	EfFgGhH				
Employee Notes		opPqQrRsStTuUvV fFgGhHiljJkKILmMr					
		WxXyYzZaAbBcCdl loOpPgQrRsStTuU					

**Note:** You will not need to change the source code, because the FETCH statement uses the Fields(*all) parameter.

2.Compile the form.

#### 3.Execute the form.

🗖 Real and Virtual Field	ls in Employee File				
Employee Number Department Code	01 ADM	Department Desc	cription	Administration	
Surname	Brown		Income ta	ix to deduct	1,250.00
Given Name	Veronica		Full Emplo	oyee Number	ADM01
Monthly Salary	5,000.00		Note Len	-	223
Start Date	1/03/1989	~		n Company	19
Notes	Et harumd dereud facilis est er exp liber a tempor cum soluta nobis elig comque nihil quod a impedit anim ik placeat facer possim omnis es volu est, omnis dolor repellend.	jend optio d quod maxim		Fetc	h

Employee ID	01	Department Description	Administration		
Surname	Green	Income	Tax to Deduct	575.00	1
Given Name	Jim		oloyee Number	ADM0	_
Department Code	ADM		ith the Company	00	
Monthly Salary	2,300.00		of Notes as Characters		70
Start Date	21/12/2012 💌				
Employee Notes	jdshakisdhakihakihdkaihdk akisdhkaishdakihdakidha	sajshdkijdhkajdhkijdhalkdh			

- a. Fetch the details for an employee.
- b. You will see the Department description such as Administration retrieved into the Department PJF field from the Department file.
- 4. Exit the form.
- 5. Close the form in the Editor.

# Step 7. Add PJFs to Department File

In this step you will add a calculation type virtual field to the Department file based on the salary data in the Employee file.

- 1. Open the Department file **iiiDepartments** in the Editor.
- 2. Ensure that the *Fields in File* tab and the *Details* tab are visible. Notice that no PJFs have been defined.
- 3. Select the Access Routes tab to add your new predetermined join fields.
- 4. Select the iii**TOEMP** access route. Click the *Add* button on the *Home* ribbon and select *Add PJF* from the drop down list.
- 5. In the *Details* tab, enter the following characteristics:

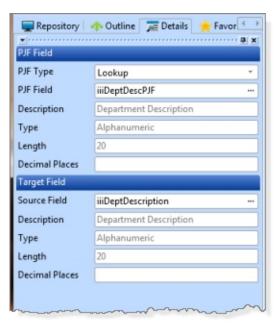
PJF field iiiTotalSalaryPJF

PJF Type Total

Source field iiiSalary

Use the ellipsis and *Find dialog* to select the fields.

Your *Details* tab will now look like this:



- 6. Compile the file.
- 7. Close the file in the editor.

# Step 8. Modify Test Form

In this step you will modify your existing *Search by Department Description* form to include the predetermined join field **Total Salaries**. This field will show the total salaries per department.

- 1. Using iiiLIST tab, locate and open form iiiSearchDeptByDesc.
- 2. Drag the field iiiTotalSalaryPJF to the form.

Department Code	ABC
Department Description	aAbBcCdDeEfFgGhHiljJ
Total Salaries	123,456,789.12
Find Department Code	

3. Switch to the *Source* tab and modify the FETCH command to either include iiiTotalSalaryPJF or change the Fields() parameter to *ALL.

Fetch Fields(*ALL) From_File(IIIDEPARTMENTS) With_Key(#IIIDEPT)

- 4. Compile your form. Check that the compile completed successfully.
- 5. Execute your form.
  - a. Fetch the Administration department (depending on what changes you made to the form in the tutorial REP006 Logical Views, you may need to fetch the department either by the description or the code).

Search by Department D	escription	
Department Code	ADM	
Department Description	Administration	
Total Salaries	6,209.00	
Find Department Code		

Notice the value for the predetermined joined field Total Salaries. It shows the combined monthly salaries of all employees in the Employee file for the department you retrieved.

- 6. Exit the form.
- 7. Close the form in the Editor.

### Summary

#### **Important Observations**

- Your file OAM needs to be recompiled once access routes are added.
- The type of PJF that can be created is based on the definition of the access route. If the relationship is one to one, only a lookup can be created. If the relationship is one to many, calculation type PJFs can be added.
- PJFs are listed in the *Field in File* tab and can be edited from this tab but they cannot be added. You need to add a PJF by first selecting an access route on which the PJF will be based.
- In addition to defining PJFs, access routes are used in LANSA templates to show related files and in LANSA Client to show linked or joined files.

# **Tips and Techniques**

- The *Keep last* parameter is used to improve performance. It specifies the number of retrieved PJF values to be kept in memory. This value applies to PJFs defined on the access route when the relationship is one to one.
- Be very careful with excessive use of PJFs. There are performance implications of overusing PJFs.
- You can also use trigger functions to perform the same type of calculations or lookup operations that are performed by a PJF.

# What I Should Know

- What an access route is.
- How an access route is defined.
- Where access routes are used.
- Where else in LANSA access routes are used.
- What a Predetermined Join Field is.
- What operations can be performed by a PJF.
- How to add and maintain PJF definitions.
- What the relationship between PJF and access routes is.

## **REP011 - Repository Summary**

#### **Objectives**

• To use the knowledge you have acquired in the previous tutorials to create LANSA fields and a file with validation rules, virtual fields, PJFs and a logical view.

The file you will create will be called Employee Holidays iiiEmpHolidays (where **iii**=your initials) with the following fields:

iiiEmployNumber	Existing employee number field	
iiiHolCode	Holiday Code	
iiiHolStartDate	Holiday Start Date	Showtime = False
iiiHolEndDate	Holiday End Date	Showtime = False
iiiHolComments	Holiday Comments	Allow lower case

iiiHolDuration	Holiday Duration (virtual field)	Edit Mask = 2
iiiSurnamePJF	Employee Surname (PJF from the employee file)	

• To create an application to maintain the file and another application to use the logical view to search the file.

To achieve these objectives, you will complete the following:

Step 1. Create the File Definition

Step 2. Modify Field Definitions

Step 3. Add Rules to the File

Step 4. Create and Execute a Test Form

Step 5. Add a Virtual Field to Your File

Step 6. Create an Access Route and a Predetermined Join Field (PJF)

Step 7. Recreate Employee Holidays Application

# Step 8. Create a Logical View and Test Summary

# Some Helpful Tips

- If you need additional help completing any of the steps, refer back to the exercises in the appropriate lessons.
- Remember to use the online Help and the LANSA documentation.

# **Step 1. Create the File Definition**

In this step you will create an employee holiday file which will contain the details of the employees' holidays.

- 1. Create a file called **iiiEmpHolidays**, Employee Holidays. Make it RDMLX enabled.
- 2. Using the Multi-Add Fields option in the file, add these real fields to the file: Don't forget to define the key fields.

iiiEmpHolidays	Employee Holiday File	Туре	Length	Key
iiiEmployNumber	Employee number	Alpha	2	1
iiiHolCode	Holiday Code	Alpha	3	2
iiiHolStartDate	Holiday Start Date	DateTime	26	
iiiHolEndDate	Holiday End Date	DateTime	26	
iiiHolComments	Holiday Comments	String	512	

# **Step 2. Modify Field Definitions**

In this step:

- 1. Change the Holiday Start Date and Holiday End Date fields to only show the date, not the time.
- 2. Change the Holiday Comments to be visualized by default as a multiline edit box, and to allow lower case

# Step 3. Add Rules to the File

In this step add rules to the Holidays file:

- 1. To ensure that the employee number in the Holidays file exists in the Employee file.
- 2. To ensure the holiday code can be either 'ANL' (Annual Leave) or 'OWN' (Leave without Pay).

Check your validation rules using the Simple Rules List in the Rules and Triggers tab.

Remember to compile your file and check your compile settings

## Step 4. Create and Execute a Test Form

In this step create a form to maintain the holiday file.

1.Switch to your Employee Holiday file in the editor and select the *File Attributes* tab. Note the Identifier for the file, for example IIIEMPHOLI. This must be used as the file name in the template.

Fields in File   Logical Views   Rules and Triggers	Access Routes   Batch Control   File Attre
	0
▲ File Settings	() ()
Name	iiiEmpHolidays
Identifier	
File Type	LANSA file
File Library	D13TRNLIB
Record Format Name	IIIEMPHOLI 🧯
I/O Module Library	Same library as file (F) 🛛 💡
File Uses SQL On IBM i	No
Alt. Collating Table	
Enabled For RDMLX	Yes
Enabled Earlong Names	Yes

- 2. Create a form **iiiMaintEmpHols Employee Holidays**.
- 3. Use the VL_BASEMNT template to create a maintenance application for the iiiEmpHolidays file.
- 4. When running the template:

a.Choose FF layout.

b.Response to "Supply a word ....." must be a single word, with no spaces. c.Include all the fields in the file.

- 5. Compile and execute the form. Make sure you are executing it locally.
- 6. Enter holiday information for employees 01, 02 and 03.
- 7. Test what happens when you try to insert information for employees that do not exist in the Employee file.
- 8. Try to use a holiday code other than ANL or OWN.

Maintain employee	Holidays
Employee Number Holiday Code	01 ANL
Holiday Start Date Holiday End date	01/09/2012
Holiday Comments	Nam tristique libero quis justo accumsan eget mollis felis dapibus. Morbi ultrices semper mollis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aenean dapibus felis vel turpis molestie nec molestie nibh sodales. Donec tortor nisl, pulvinar sit amet tincidunt ornare
<u> </u>	Insert Update Delete

#### Step 5. Add a Virtual Field to Your File

In this step create a field Holiday Duration iiiHolDuration which will be used as a virtual field in the Holiday file.

Field Name	Field Description	Type	Length	Virtual
iiiHolDuration	Holiday Duration (virtual field)	Alpha	40	Yes

1. Create the virtual field derivation in your file. It will show the holiday dates like this:

Holiday Duration	19th MARCH 2008 - 17th JANUARY 19

2. Make it a *Code Fragment* derivation. It needs to be derived when the record is read.

Copy this code from the online guide Tutorials to create the field derivation:

#iiiHolDuration := #iiiHolStartDate.AsDisplayString(
DDXXbMMMMMMMMbCCYY ) + ' - ' +
#iiiHolEndDate.AsDisplayString( DDXXbMMMMMMMbCCYY )

**Note:** Changing your font to a small size such as 8pt (see *Options / Font*) will make editing the above code much easier.

# **Step 6. Create an Access Route and a Predetermined Join Field** (PJF)

In this step create an access route from the Employee Holidays file to the Employee File. The number of records is 1.

1. Create a field for the employee surname to be used as a PJF as follows.

Field Name	Field Description	Type	Length	
iiiSurnamePJF	Employee Surname	Alpha	25	Allow LC

- 2.Define an *Access Route* to the Employee file, with a key of iiiEmployNumber.
- 3. Define the *Predetermined Join Field* to retrieve iiiSurnamePJF from the employee surname (iiiSurname) in the Employee file. This is a lookup PJF. Remember to compile your file.

## **Step 7. Recreate Employee Holidays Application**

In this step recreate the Employee Holidays application again to include the virtual field and the predetermined join field.

- 1. Open iiiMaintEmpHols and display the Source tab.
- 2. Delete all the code in the Source tab.
- 3. Run the template VL_BASEMNT and include all the fields.
- 4. Make sure the Holiday Duration and the Employee Surname fields are not hidden behind the Holiday Comments field on the form.
- 5. Compile and execute your form.
- 6. Fetch holiday details for an employee and holiday code.

Employee ID	01
Holiday Code	ANL
Holiday Start Date	03/12/2012
Holiday End date	28/12/2012
Holiday Comments	
Holiday Duration	3rd December 2012 · 28th December 2012
Sumame	Green

Note the Holiday Duration and Employee Surname fields.

# Step 8. Create a Logical View and Test

In this step first create a logical view of the Employee Holidays file keyed by the Holiday Code:

- 1. Create a logical view called iiiEmpHolByCode and key it by the iiiHolCode field.
- 2. Compile the file. Use *Verify / Compile* to check you are using the required compile options.

Create a form to test the logical view:

- 3. Create form iiiEnqEmpHols with the description Holidays by Holiday Code:
  - a. Display the *Common Controls* tab and drag a **Push button** and a **List view** to your form.

_ <b>D</b> _ X _
111
111
111
111
111
<u></u>

- b. Change the button *Caption* to **Fetch**.
- c. Select the *Events* tab and double click on the *Click* event to create an event routine for the push button.
- d. Display the *iiiLIST* tab, expand the file iiiEmpHolidays and drag the iiiHolCode (Holiday Code) to the top of the form.
- e. Drag the Employee Number (iiiEmployNumber), Surname (iiiSurnamePJF) and the Holiday Duration (iiiHolDuration) into the list view. You will probably need to widen the list view and the columns which will have been created for the fields.

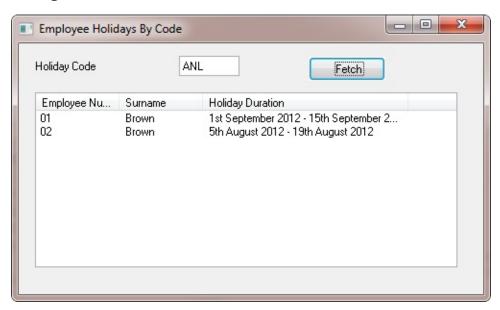
- f. Display the *Source* tab and complete the push button Click event routine.
- Clear the list view.
- Select entries from the logical file in holiday code order.
- Retrieve each entry with the key Holiday Code.
- Add an entry to the list view.
- End the Select loop.

New code is shown in red.

```
Evtroutine Handling(#PHBN_1.Click)
Clr_list #LTVW_1Select Fields(*ALL) From_File(iiiEmpHolByCode)
With_Key(#iiiHolCode)
Add_Entry To_List(#LTVW_1)
Endselect
Endroutine
```

Replace the file and field name with your own holiday file name and holiday code name.

- g. Compile and execute the form.
- h. Enter a holiday code and press the button to fetch the records with matching code:



i. Replace the Holiday Code field on the form with the Employee Number field.

j. Switch to the Source tab and change the Select statement to:Select Fields(*ALL) From_file(iiiEmpHolidays)With_key(#iiiEmployNumber)

The application will now retrieve records from the Holiday file based on the employee number:

🗈 Employee Holidays By Employee		
Fetch		
Holiday Duration		
1st September 2012 - 15th September 2012 7th May 2012 - 14th May 2012		

k. Close the form.

# Summary

# **Tips and Techniques**

• LANSA provides a logical modeling tool that can help you to quickly build new databases or extend existing databases.

# What I Should Know

• How to use the features in the LANSA Repository to create an application database.

# **REP012 - Check In Objects (Optional)**

### **Objective:**

• To learn how to check in objects to the LANSA Master Repository.

In order to complete this exercise:

- You must have completed the previous exercises.
- You must have a properly installed and configured Visual LANSA Slave System with a working connection to the LANSA Master System (a System i is assumed in this exercise). If you are working on an Independent LANSA system you cannot do this exercise.
- You need to be a licensed Visual LANSA developer and you need the proper LANSA licenses installed that allow both server and Visual LANSA development. You cannot complete this exercise if you are using a trial (unlicensed) version of Visual LANSA.

In this tutorial you will check into the master system the Department Maintenance application you created in the previous tutorials.

To achieve these objectives you will complete the following:

Step 1. Confirm Connection to LANSA Master System (Optional)

Step 2. Check in the Department File and Fields

Step 3. Verify Objects Exist on the Server

Step 4. Execute Your Application Client Server

Summary

# **Before You Begin**

You may wish to review the following topics: In the *Visual LANSA User Guide*:

- Check In Tab
- Check Out Tab
- Propagation Tab
- In the Visual LANSA Administrator Guide
- Host Monitor.

# Step 1. Confirm Connection to LANSA Master System (Optional)

You can only perform this step if you can access your LANSA master system on the iSeries.

In this step you will check the contents of your LANSA master system and confirm that your master system profile and task ID are valid by connecting to the master.

1. Logon to your LANSA master system using Client Access or any available 5250 emulator. Use your developer profile and task ID to confirm that the profile and task ID are valid for development on the master. Logon to the LANSA partition where you will check in your changes. For example:



Note: Use the partition name assigned for training.

2. From the LANSA Main System Menu, select the option to *Work with Files*.

Use the *Position to* field, to list files beginning with your initials. At this point your files are not defined in LANSA for iSeries. Your list of files will look something like the following:

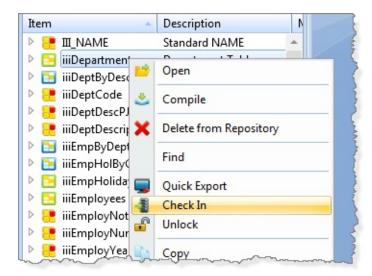
0pt	File	Library	Description		Act	Lts	Lgls	
<u> </u>	LW3FRM05	D13TRNLIB	L4Web Frameworks - Proc/Func X-Ref		2	2	0	
-	PSLEVENT	D13TRNLIB	Personnel Event Log	×	1	1	2	
<u> </u>	PSLIMG	D13TRNLIB	Personnel Images	×	1	1	0	
<u> </u>	PSLMST	D13TRNLIB	Personnel		2	2	2	
<u> </u>	PSLSKL	D13TRNLIB	Personnel skills		2	2	0	
·	PSLTIMES	D13TRNLIB	Personnel time sheets	×	1	1	2	
·	SECTAB	D13TRNLIB	Section code table		2	2	0	
<u> </u>	SKLTAB	D13TRNLIB	Skill code table		2	2	Θ	+

3. Exit the LANSA master system.

### Step 2. Check in the Department File and Fields

In this step you will check in your Department File (iiiDepartments) and its fields. When you check in a file you should always consider whether there are any fields or system variables which it depends on, which need to be checked in at the same time.

- 1. Display your dynamic list iiiList on the **Favorites tab**. It contains all fields, files and forms beginning with your initials iii.
- 2. Select the file iiiDepartments and right click to display the context menu.
- 3. Choose the check In option



4. In the *Check In* dialog, expand the *Files* node and select the file iiiDepartments.

Check in Options	
×	
*	File check in options
<ul> <li>Files (1)</li> <li>iiiDep Department Ta</li> </ul>	<ul> <li>Compile file</li> <li>Compile only if necessary</li> <li>Rebuild table</li> <li>Rebuild indexes and views</li> <li>Rebuild OAMs</li> <li>Strip debug information</li> <li>Delete \$\$ file</li> <li>Produce source listing</li> <li>Ignore Decimal Data Error</li> </ul>
🗖 Keep Locks	OK Cancel

5. Click on the *Cross References* button en on the toolbar to include the fields used by this file.

The *Local Cross References* dialog is displayed. This shows all objects referenced by file iiiDepartments and its dependents.

The file must be compiled locally to generate the Local Cross Reference information.

Local Cross References			
Name	Description	Allow check-in	Qualifier
🔁 iiiDepartments	Department Table	$\checkmark$	D13TRNLIB
<ul> <li>▷ ■ *BLANKS</li> <li>▷ ■ *ZERO</li> <li>▷ ● @@UPID</li> <li>▷ ● iiiDeptCode</li> <li>▷ ● iiiDeptDescription</li> <li>▷ ● iiiEmployees</li> </ul>	Blank / blanks variable Zero (0) variable Field update / access identifier Department Code Department Description Employee File	**	D13TRNLIB
		•	Close

You do not need to check in system variables and standard fields which will already be defined in the master repository.

Notice that the *Allow Check-in* column highlights the objects which could be checked in.

- 6. Select the fields iiiDeptDescription, iiiDeptCode and iiiTotalSalaryPJF.
- 7. Click on the *Add for check* in button 🖶 to add these fields to the check in.

The *Check In* Options dialog now shows all the selected objects. Note that the check in options available depend on the type of objects being checked in.

🔹 Check in Options					
<ul> <li>Files (1)</li> <li>iiiDep Department Ta</li> <li>Other LANSA</li> <li>iiiDept Department Co</li> <li>iiiDept Department De</li> </ul>	File check in options         Compile file         Compile only if necessary         Rebuild table         Rebuild indexes and views         Rebuild OAMs         Strip debug information         Delete \$\$ file         Produce source listing         Ignore Decimal Data Error				
🔲 Keep Locks	OK Cancel				

In this case you are checking a new file to the master repository. The *Compile File* option shown, will build table, indexes and OAM for the new file.

When checking in a changed file you should select the appropriate rebuild options.

Delete \$\$ file is required when rebuilding a file, so that LANSA can rename the existing file and map data from old file to new.

Depending on the user id and task tracking settings in your system, a Keep Locks option may appear on the bottom left of this dialog. Selecting this option ensures that any new or modified objects checked into the master will remain locked to the Task ID you are using. For further information, refer to Unlock Objects in Task Tracking in the *Administrator's Guide*.

8. Press *OK* to start the check-in.

The Check In tab shows the progress of the check in.

×		Job Status	Description	Results	Currently Processing	Started 👻	Ended
× 루.		Completed	Check in 3 objects	0 fatal errors - 0 warnings	EARTH13GA - RDML Com	11/12/2012	11/12/2012
	5						
P 💊	< L						
📩 Co	📩 Compile 🔄 Check In 📲 Check Out   📓 Propagation   🤪 Help   🍖 Web Designs						
Ready						LANS	SA13 TRN

9. Wait for the check in to complete and verify no errors occurred.

-	Job Status	Description	Results	Curren
	Completed	Check in 3 objects	0 fatal errors - 0 warnings	EARTH
×	Objects			
~	🔚 iiiDeptCode	Department Code		
×	📑 iiiDeptDescription	Department Description		
	🔁 iiiDepartments	Department Table		

10.Notice that with your check in selected in the *Check In* tab, you can click the *View Detailed Messages for this job* icon *local to see a more detailed log.* 

	Date	Time	Message
V EARTH13GA		Completed	
V Definitions		Completed	
▲ ✓ RDML Compiles		Completed	
72	2012-12-11	16:18:06	Copying Package to /LANSA_d13pgmlib/x_lansa/x_hmrgst
73	2012-12-11	16:18:06	Request to install Application Package 'A7800111D0000003' sent to the Hi
74	2012-12-11	16:18:06	Request to compile File 'D13TRNLIB/IIIDEPARTM' sent to the Host Reposi
86	2012-12-11	16:19:03	Job 751057/JIVORV13/U0000003 submitted to job queue QBATCH in librar
0, 88	2012-12-11	16:19:03	Unit of work U0000003/A7800111 has started processing object A7800111/
0, 89	2012-12-11	16:19:03	The job attributes changed to QOTHPRDOWN
90	2012-12-11	16:19:03	Error detected when attempting to Open or Write to or Read from file /LA
0, 91	2012-12-11	16:19:03	Installation of Package - D0000003 - Started
0, 92	2012-12-11	16:19:03	Copy of file IIIDEPARTM.ctd was successful
93	2012-12-11	16:19:03	Copy of file iiidepartm.ctd was successful
94	2012-12-11	16:19:03	Installation of Package - D0000003 - ended successfully
95	2012-12-11	16:19:03	Library list changed.
96	2012-12-11	16:19:03	The job attributes changed to JIVORY13
0, 97	2012-12-11	16:19:03	Unit of work U0000003/A7800111 has started processing object IIIDEPART.
98	2012-12-11	16:19:05	Object IIIDEPARTM in library D13TRNLIB not found.
99	2012-12-11	16:19:05	File SQLTMP created in library QTEMP.
0, 100	2012-12-11	16:19:05	Member SQLTMP added to file SQLTMP in QTEMP.
0, 101	2012-12-11	16:19:05	Object SQLTMP in QTEMP type *FILE deleted.
0, 102	2012-12-11	16:19:05	FILE definition IIIDEPARTM from D13TRNLIB has now been unlocked
0, 103	2012-12-11	16:19:05	Create or re-create of file IIIDEPARTM from
0, 121	2012-12-11	16:19:09	Compile transactions for remote system monitor job 000000003 have com
RDMLX Compiles		Completed	

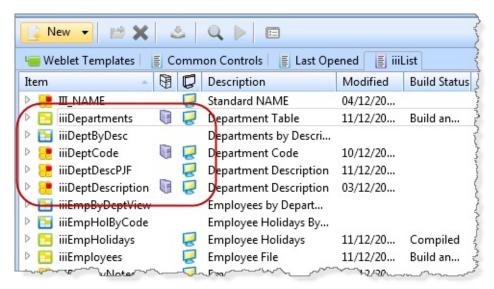
11.Notice that with any of the log detail lines beginning *Q* 7298 you can click the **Show IBM i joblog** icon *Q* to show the joblog from the server.

🔍 QPJOBLOG	- 184412/JIVORY12/U000025	3
💾 Find		A A Page
5761SS1 V	V6R1M0 080215	Job Log
	e cription	U0000253 User JIVORY12 QBATCH Library QGPL
MSGID	TYPE	SEV DATE TIME FROM PGM LIBRA
CPF1124	Information	00 01/06/11 16:51:25.418629 QWTPIIPP QSYS Message Job 184412/JIVORY12/U0000253 16:51:25 in subsystem QBATCH in QSYS. Job ente 16:51:25.
CPI1125	Information	00 01/06/11 16:51:25.418954 QWTPCRJA QSYS Message Job 184412/JIVORY12/U0000253 Cause Job 184412/JIVORY12/U0000253 in QGPL from job 184134/JIVORY12/U0000253 was started using the Submit Job (SEMJOB) comm attributes: JOBPTY(5) OUTPTY(5) PRTIXT() RTGDT QSYS2 QHLPSYS QUSRSYS) CURLIB(*CRTDFT) D12DTALIB D12DEMOLIB QTEMP D12JSMLIB QG *SECLVI LOGCLPGM(*NO) LOGOUTPUT(*JOBEND) OUTQ INQMSGRPY(*RQD) HOLD(*NO) DATE(*SYSVAL) SWS(00 MSGQ(QUSRSYS/JIVORY12) CCSID(65535) SRTSEQ(*N/ JOBMSGQMX(32) JOBMSGQFL(*WRAP) ALWMLTTHD(*NO)
*NONE	Request	01/06/11 16:51:25.419952 OWTSCSBJ MessageCALL PGM(D12PGMLIB/LANSA) PA 'DEM' 'ENG

# Step 3. Verify Objects Exist on the Server

In this step you will review how Visual LANSA shows objects that exist in the master repository on the server.

- 1. Display the *iiiList* tab on the *Favorites* tab.
- 2. Right click on any column header and ensure that the *Local Repository State* and *Master Repository State* are shown..
- 3. Widen the iiiList tab and / or drag the *Local and Master Repository State* columns into view if necessary.



Notice that the check in of file iiiDepartments and its fields has refreshed this column. The server icon shows that the objects now exist in the server repository.

The Master Repository State column will be blank if the object is not yet defined in the master repository.

### **Step 4. Execute Your Application Client Server**

In this step you will execute your Department Maintenance application in client server mode, to access the new Department Table which you have just created on the server.

- 1. In the iiiLIST tab select the form iiiMaintDepartment and click on the Execute ▶ button.
- 2. In the Execute dialog select the appropriate server type, for example Form as Client to RDMLX iSeries Server.

Execute					
Form as Client to any Server (manual connection) Form as Client to Linux Server Form as Client to RDML IBM i Server Form as Client to RDMLX IBM i Server Form as Client to Windows Server Form as Windows Application Form using DirectX					
Prompt for additional execution parameters					
OK Cancel Help					

3. Press OK. Your application starts executing connected to the server.

Maintain Departments	
Department Code	I
Department Description	
<u>F</u> etch	nsert <u>U</u> pdate <u>D</u> elete

In this case your form is accessing the Department Table on the server via its OAM which was also created when you checked the file in.

4. Try to fetch the details of the department **ADM**.

The Department will not be found because no data was checked in when the file definition was moved to the server. (Note that you can create simple functions to transfer data between LANSA master system and Visual LANSA files.)

- 5. Insert the details for the Administration department and then *Insert* them.
- 6. Try to insert another department with the code and description blank. Note that you see the same validation error messages as when you tested the form locally. The OAM on the server, has the same business rules compiled into it.
- 7. Exit your form.

#### Summary

#### **Important Observations**

- Check in options allow compilable objects to be compiled when checked in to the server.
- To ensure that all required objects are checked into the server, use the cross references facility in the *Check In Options* dialog.
- If you are using task tracking, refer to Using Task Tracking in LANSA in the *Visual LANSA Administrator Guide*.
- Display the Master Repository State column for objects to see if they exist on the server.

# **Tips and Techniques**

• Review Rules for Repository Synchronization in the *Visual LANSA Administrator Guide*. It is an efficient means of keeping your Visual LANSA systems current.

### What I Should Know

• How to check in objects to the LANSA Master Repository.

# **Programming RDML with Visual LANSA Forms**

# What is the Programming Tutorial?

This tutorial is a series of exercises that are designed to introduce the fundamental skills required to begin programming with RDMLX using Visual LANSA Editor with graphical form-based development.

The following exercises are included:

FRM015 - Getting Started with Forms Programming

FRM025 - Insert a Database Record

FRM035 - Maintain a Simple Database Table

FRM035 – Appendix

FRM055 - List Component Basics

FRM065 - Using List Components

FRM075 - Using a Working List

FRM085 - Update from a Grid

FRM095 - Calling a Function

FRM105 - Define a Trigger Function

FRM115 - Writing Reports

FRM125 - Check Out / In to IBM i

# Who Should Use the Tutorials?

These tutorials have been written for new LANSA developers. They introduce basic skills required when creating graphical form-based applications or WAM, Web Function and Integrator applications.

# **Before you Begin**

You must have LANSA Demonstration Personnel System installed in the partition that you will use with the set up options as described in What Partition Should I Use?.

The LANSA Demonstration Personnel System contains all the objects used by these exercises.

You should have completed the

- Visual LANSA User Interface Tutorials
- LANSA Editor Tutorials

• Visual LANSA Repository Development Tutorials.

### Tips for using the exercises:

Check off each step in the exercise as you complete it.

Follow the instructions very carefully.

Remember to replace iii with your unique 3 characters. You will not always be reminded to make this substitution. For further information refer to How many developers can use the exercises?

These exercises assume that you have not previously customized the editor interface. If you have already customized your environment, the example screens and instructions may not exactly match your customized development environment.

The following are important notes regarding the structure of the exercises:

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

Later exercises are designed to use skills from the earlier exercises. These exercises are designed to be completed in sequence.

# **FRM015** - Getting Started with Forms Programming

#### Introduction

The Hello World exercise is an introduction to the Visual LANSA editor. You will develop a simple form-based application and then you will add further components and functionality and explore programming using events, properties and methods.

Hello
World
Clear
Close

#### **Objectives:**

- To introduce the basic concepts of components, properties, events and RDML commands.
- To learn how to add events to a form.
- To learn how to edit properties of components.
- To learn how to use methods.
- To learn how to compile and execute your form.

To achieve these objectives, you will complete the following:

Step 1. Editor Settings

- Step 2. Create a Component
- Step 3. Add Components to the Form
- Step 4. Change the Properties of a Component
- Step 5. Add Remaining Push Buttons and Set their Properties
- Step 6. Add a Field to the Form and Set its Properties
- Step 7. Create Event Routines for the Push Buttons

Step 8. Add Logic to the Hello Button Click Event
Step 9. Add Logic to the Other Click Events
Step 10. Compile the Form
Step 11. Execute the Form
Step 12. Align and Size Components
Step 13. Component Definitions
Step 14. Understanding Events
Step 15. Using Component Properties
Step 16. Understanding Component Methods
Summary

### **Before You Begin**

You may wish to review the following topics:

- Editor Features in the Visual LANSA User Guide.
- Component Concepts in the Technical Reference Guide.
- The Component Model in the Visual LANSA Developer Guide.

# **Step 1. Editor Settings**

In this step you will logon to Visual LANSA and set the editor options

- 1. Logon to Visual LANSA. If you cannot remember how to do this, refer to the LANSA User Interface Tutorials. Once started, the Visual LANSA Editor is displayed. The appearance of the editor will depend upon the type of installation as well as the editor settings.
- 2. To change the editor settings, choose Editor *Options* from the File tab.

🔽 l 🐸 🖪 迭 🔶 🕫	LANSA Editor
File	
ave Save	Last Opened
Close	
📂 Open	
Last Opened	
New	
🗐 Save All	
🚔 Print	
Close All	
System Information	
🙀 Options	
🔀 Exit	

It is recommended that you set the *Source* settings to include *Component Definitions*. These settings are included in the exercise *Format Source Code* in the *LANSA Editor Tutorials*.

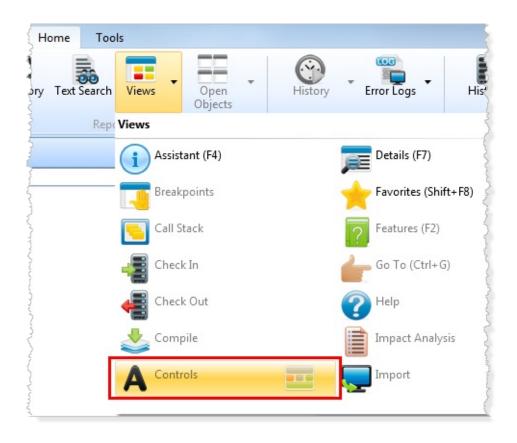
Remember that you can also turn on the *Auto Hide* tab feature to make more space to view source code in the editor. For more details, refer to exercise VUI002 Editor Parts in the *User Interface Tutorials*.

3. Make sure you can view the *Controls* tab. This tab contains all the controls frequently used on a component. In V13 SP1, this is now a separate tab which

looks like this:



If the *Controls* tab is not open, *open it from the Home ribbon*, *Views menu:* 



## Step 2. Create a Component

In this step you will create a Form and open it in the LANSA editor.

1. On the *File* menu, use the *New* button and select *Form / Basic Form*.

The *Create form* dialog is displayed.

Enter the following for the form:

- a. Long Name: iiiCOM10 (where iii=your initials) *
- b. Description: Hello World
- c. Enable for RDMLX (Yes)

* If you are using a trial version of LANSA, use iii=DEM (DEMCOM01). If the form already exists, it needs to be deleted first.

d. Press the *Create* button.

New Form		×
Name	iiiHelloWorld	Create
Description	Hello World	Cancel
Framework	Personnel & Payroll (HUMAN RESOURCES)	Curren
Group		
Identifier	IIIHELLO	
Enabled For RDMLX		

When you create a component you should select a suitable *Framework*, such as Personnel & Payroll (HUMAN RESOURCES). One way to find a components on the *Repository* tab, is to look under *Organizers / Frameworks / HUMAN RESOURCES*. You can change a component's Framework at a later stage.

2. The form will be opened in the editor.

# Step 3. Add Components to the Form

In this step you will insert a standard push button control to the form.

1. The *Design* tab, showing the form layout, is selected by default. This layout shows how the form will appear at execution time. It is used to design and build your application.

Size the form by clicking on the corner of it, keeping the mouse button down and dragging the corner to make it about this size:

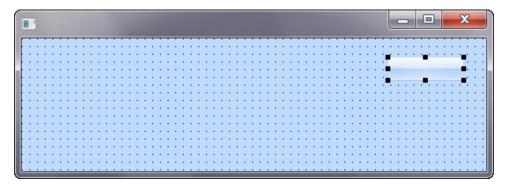
																																																	]			X	3	
	d																																											-										
-																				-		_					-	-												_							_		-		-	-	-	_
																												•																				•						
•					•	•	•		•		•	•		•		•	•	•	•											•		•	•					•	•			•	•		•	• •			· •	÷.,				
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	• •	• •			•	•	•	•	•	•	•	•	•	•	• •	•	•	•	•	•	•	•	•	•	•	• •	•	•	•	•	•	• •	• •	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	• •		•	•	•		• •		· .
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	• •				•	•	•	•	•	•	•	•	•	•	• •	•	•	•	•		•	•	•	•	•			•	•	•		• •		•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	• •	• •		• •	• •	•	•	•	•	•	•	•	•	•	• •	•	•	•	•		•	•	•	•	•	• •	• •	•	•	•	•	• •	• •	•
•					•	•	•	•		•	•	•	•	•	•											•	•	•	•	•	•				•	•	•						•					•	•	•				
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	• •	• •	• •	• •	•	•	•	•	•	•	•	•	•	• •	• •	•	•	•	• •		•	•	•	•	•	• •	•	•	•	•	•	• •	• •	•
•					•						•		•														•	۰.		•							•											•	÷ • •	•				
																											Ξ.	÷.,																				÷.,	÷.,					
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	• •	• •	• •	• •	•	•	•	•	•	•	•	•	•	•	• •	•	•	•	•		•	•	•	•	•	• •	• •	•	•	•	•	• •	• •	•
																												•																										
•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•								•	•	•	•	•	•	•	•		•	•	•	•		•	•	•	•	•	• •			•			• •		

- 2. Add a push button to the form:
  - a. Display the *Controls* tab.
  - b. Locate the *Push button* control.
  - c. Drag and drop the Push Button control onto your form.

To drag and drop, left click on the push button control in the list and hold the left mouse button down. Move the cursor over to the form layout and release the mouse button to drop the control at the cursor position.

Alternatively double-click the push button control in the list to add it to the form and then drag it to the right.

Your form layout should now appear something like this:



Note: You can also use *Ctrl* + *Cursor Key* to move any component around on the form.

### **Step 4. Change the Properties of a Component**

In this step you will change the *Caption* and the *Name* properties of the push button.

1. Double-click the push button to display the *Details* tab.

The *Details* tab is used to change the *Properties* of a control, or create an *Event* handling routine or a *Method* routine for it.

- a. Change the *Name* property of PHBN_1 to Hello.
- b. Change the *Caption* property to Hello.

Now you can see the caption *Hello* added to the push button on the form.

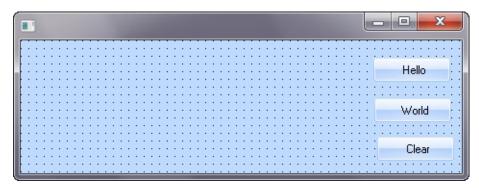
### **Step 5. Add Remaining Push Buttons and Set their Properties**

In this step you will add the World and Clear push buttons to the form.

1. Using the *Common Controls* tab, double-click the Push button control twice to add two more buttons to the form.

Position the push buttons under the first button. Do not worry about aligning the push buttons accurately at this stage. You will learn how to do this in a later step.

- 2. Using the *Details* tab, name the push buttons *World* and *Clear* and apply appropriate captions to them just as you did in the previous step.
- 3. Your form should appear something like this:



### Step 6. Add a Field to the Form and Set its Properties

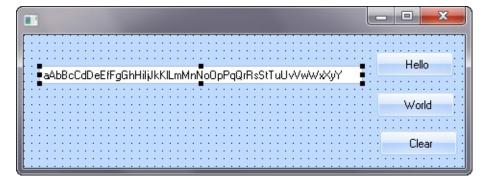
In this step you will add a field from the LANSA Repository to your form to display the *Hello World* text. You will change the field margins so that you don't display the field label.

- 1. On the *Repository* tab, select the *Fields* node.
- 2. If Alphabetical Groupings is on, expand the letter *S* to see the list of fields starting with the letter S.
- 3. Locate the field *STD_TEXT*.
- 4. Drag and drop the *STD_TEXT* field onto your form.
- 5. Double-click the field to display the *Details* tab and change the *MarginLeft* property of the field to 0 so that the label is not visible.

You may wish to reposition the field on the form once the label is no longer displayed. You may also adjust the *Width* of the field using the *Details* tab, or by using the *Design* tab to resize the field with the mouse.

Note: with the field selected, you can move the field using the cursor keys while holding down the Control key. This is often the easiest way to position a component accurately.

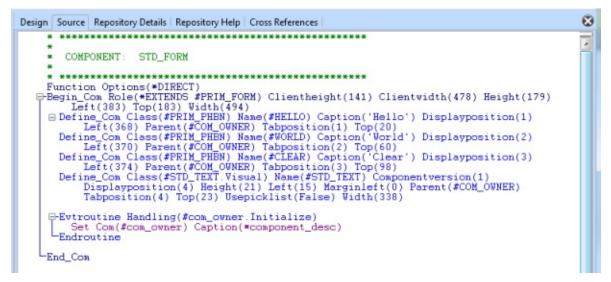
Your form should appear something like this:



### **Step 7. Create Event Routines for the Push Buttons**

In this step you will add event routines for each push button. An event routine is a routine that is invoked when the user or the program triggers a specific action. You will add a event handling routine to be called when the user *Clicks* on each push button.

1. Select the *Source* view in the Editor. You should see code similar to the following:



For the moment just ignore the existing code.

- 2. Next create an event routine (EVTROUTINE/ENDROUTINE command pair) for the *Clear* button to handle the 'Click' event:
  - a. Select the *Design* tab and select the *Clear* button.
  - b. Right-click, select the *Events:Push Button* option and then the *Click Event*.
  - c. A new event routine is added for the *Clear* push button in the *Source* tab as shown below:

Notice the format of the event routine names. Event routine names are always formatted as

COMPONENT.EventName (#Clear.Click).

3. Next add a Click event to the *Hello* and *World* buttons.

Your event routines should now look like this:

Evtroutine Handling(#HELLO.Click) Endroutine Evtroutine Handling(#WORLD.Click) Endroutine Evtroutine Handling(#CLEAR.Click) Endroutine

#### Step 8. Add Logic to the Hello Button Click Event

1. Add the line of code shown, highlighted inred, italic to the *Click* event for the Hello button (it adds the word 'Hello ' to the #STD_TEXT field):

EVTROUTINE HANDLING(#HELLO.Click) #STD_TEXT:= #STD_TEXT + 'Hello ' ENDROUTINE

The AutoComplete prompter shows up as you start to type. You can use it to select the command and the parameter values.

Note that there is a space between the word *Hello* and the closing quote.

### Some Things to Note About Editing Code

- The editor is completely free format. The syntax will be checked as you enter the command. A red triangle beside a command indicates that a warning or error message can be displayed. To display the message, click on the red triangle.
- You can only enter one command per line.
- Your commands and their parameters can be in uppercase or lowercase.
- The definitions of the components (that is, buttons, fields and so on) on your form are stored at the start of your program as a series of DEFINE_COM (Define Component) commands. These commands can be hidden or displayed using the plus sign in front of them or using the editor settings.
- Normally, you will change the DEFINE_COM commands only by using the form layout or by altering the component's property sheet even though you can edit the DEFINE_COM statements.

# Step 9. Add Logic to the Other Click Events

In this step you will add the required code for each of the Push Button event routines.

When you are finished with this step, you will have entered the three lines of RDML code implement the action required by each the push button:

*When the user clicks the *Clear* push button, the contents of the STD_TEXT f changed to blanks. EVTROUTINE HANDLING(#CLEAR.Click) Change Field(#STD_TEXT) TO(*BLANKS) ENDROUTINE *When the user clicks the *Hello* push button, the word *Hello* is concatenated tc contents of the STD_TEXT field. EVTROUTINE HANDLING(#HELLO.Click) #STD_TEXT := #STD_TEXT + 'Hello ' ENDROUTINE *When the user clicks the *World* push button, the word *World* is concatenated t contents of the STD_TEXT field. EVTROUTINE *When the user clicks the *World* push button, the word *World* is concatenated t contents of the STD_TEXT field. EVTROUTINE HANDLING(#WORLD.Click) #STD_TEXT := #STD_TEXT + 'World ' ENDROUTINE

- 1. Type the CHANGE command for the *Clear* button *Click* event: Change Field(#STD_TEXT) To(*BLANKS)
- 2. Put the cursor on the CHANGE command in the editor and press the F1 key to display the online help to review specific technical details about using this command:
- 3. Complete the World.Click event routine by entering the following statement:
  #STD_TEXT := #STD_TEXT + 'World '
- 4. Put the cursor anywhere on the STD_TEXT statement in the editor and press the F1 key to display the online help to review specific technical details about using the ASSIGN command:

Your finished code should look like this:

```
-Evtroutine Handling(#HELLO.Click)
    #STD_TEXT := #STD_TEXT + 'Hello '
    Endroutine
-Evtroutine Handling(#WORLD.Click)
    #STD_TEXT := #STD_TEXT + 'World '
    Endroutine
-Evtroutine Handling(#CLEAR.Click)
    Change Field(#STD_TEXT) To(*blanks)
    Endroutine
-End_Com
```

There should be no red triangles in the source code which indicate an error. If any errors exist, they must be corrected.

Note that in the event-driven program model, the order of the event routines in the code is not important to the execution of the form.

5. Click on the save toolbar icon to save the form.

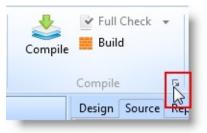


### Step 10. Compile the Form

In this step you will compile your new form.

Your components are compiled from within the LANSA editor by using the *Compile* button on the *Home* ribbon. Objects can also be compiled using context menus from various tabs in which case the compile options will simply default to the last values set in the *Compile options* dialog. Note that for this last compile method you must save your RDMLX source first.

1. Click the dialog box launcher in the *Compile* group in the ribbon to display the *Compile options* dialog.



- 2. Press OK to submit the compile.
- 3. Using the *Compile* tab at the bottom of the editor, you can review the compile status.

🗵 🔎	Job Status	Description	Results
<b>#</b>	Completed	iiiHelloWorld - Hello World	Compiled 1 of 1
1 m			
X			2
• ×	4		
( Assis	stant  😃 Compile		
			LANSAB13   TRN   JIVC

- 4. Once the compile has completed, double-click on the status message to display the compile message window. If errors have occurred, you can review the messages in this window.
- 5. Close the Compile messages window.

### Step 11. Execute the Form

In this step you will execute the form.

Your components can be executed from within the LANSA editor by using the *Execute* button on the *Home* ribbon. Objects can also be executed using context menus from various tabs or by pressing Ctrl + Shift + E.

- 1. Click the dialog box launcher in the *Execute* group in the ribbon. The *Execute*... dialog will be displayed.
- 2. Press the *OK* button to execute the form.
- 3. Check that your form functions correctly:
  - a. Click the *Hello* button. The word 'Hello ' should be added to the field.
  - b. Click on the *World* button. The word 'World ' should be added to the field.
  - c. Click the *Clear* button. The field should be cleared.

#### Step 12. Align and Size Components

1. With the *Design* view open for form **iiiHelloWorld**, select the top push button, hold down the shift key and select the other push buttons.

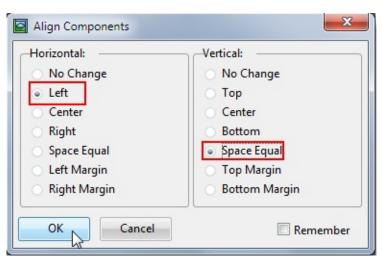
Note the 'handles' for the first button are black. The handles for the other two buttons are white. You are going to align the second and third button relative to the first.

}	
} 	Hello
	World
}	Clear

2. Display the *Design* tab on the ribbon and select the *Align* command.



3. Select the *Horizontal / Left* and *Vertical / Space Equal* options:



Note: The Remember checkbox will retain these settings for this session of

Visual LANSA

Your form should now look like this:

- Hello World Clear
- 4. Select the Hello button and reduce its height and width.

**Tip:** First unselect the group of buttons by clicking anywhere on the form.

- 5. Resize the Hello button by holding down the left mouse button on one of the handles and dragging. Alternatively, select the button and hold down the Shift key and using the cursor control keys.
- 6. Again, holding down the Shift key, select the Hello button and then the other two buttons.
- 7. Use the Edit / Size dialog to resize the second and third button, based on the first.

aAbBcCdDeEfFgGhHiTjJkKILmMnNo	oOpPqQrRsStTuUvVwWxXyY	Hello
Size Components Horizontal: No Change Same Width	Vertical:	Clear
OK Cancel	Remember	

8. Select the buttons to be the same width and same height.

Your form should look like the following:

{	
	Hello
	World
}	
{ <u></u>	· · · · · · · · · · · · · · · · · · ·

### **Step 13. Component Definitions**

1. Click on the form title bar, and select the *Details* tab on the left (or press F7). The *Details* tab is displaying the form properties:

Details							
iiiHelloWorld	Ŧ	4					
Properties Events Met	hods						
E • ?							
BorderIcons	Maximize+Minimize+SystemMer						
Caption							
ClientHeight	141						
ClientWidth	478						
Cursor	*NULL						
PragStyle	None						
EnableChildren	False						
🖻 Enabled	True						
🖻 EnsureVisible	Application						
FormPosition	Designed						
PormStyle	Normal						
FrameStyle	Sizable						
Class 🧖	*NULL						
🖻 Height	179						
🖻 Hint							
PintPopup 🖉	*NULL						
HintShow	True	E					
P HintTitle							
🖉 Icon	*NULL						
🖻 Image	*NULL						
ImageAlignment	Center						
Lavorthorage	MANULL						

For example, note the *Height* property is highlighted.

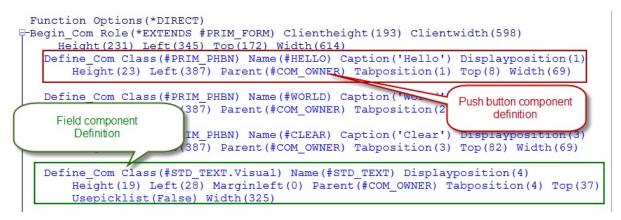
- 2. Resize the form and note the change in this property.
- 3. Select the Source tab.

Design	Source Repository Details Repository Help Cross References
00001	* *************************************
00002	*
00003	* COMPONENT: STD_FORM
00004	*
00005	* **************
00006	FUNCTION OPTIONS(*DIRECT)
00007	■ BEGIN_COM ROLE(*EXTENDS #PRIM_FORM) CLIENTHEIGHT(146) CLIENTWIDTH(519)
	HEIGHT(184) LEFT(403) TOP(210) WIDTH(535)
00008	<pre>DEFINE_COM CLASS(#PRIM_PHBN) NAME(#Hello) CAPTION('Hello')</pre>
	DISPLAYPOSITION(1) HEIGHT(34) LEFT(416) PARENT(#COM_OWNER)

The component source is defined within the Begin_Com / End_Com statements.

The Begin_Com includes the form properties. Note that Height is highlighted. As usual, RDML / RDMLX does not display properties or command parameters, which have default values.

4. Examine the Define_Com component definitions.



Most components are defined at the top of the source code.

- 5. Select the *Design* view. Select the field STD_TEXT. Note that the *Details* tab shows the field component properties. Once again the Define_Com code shows only properties which have non-default values.
- 6. With your component definitions currently shown as follows:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_FORM) Clientheight(193) Clientwidth(598)
Height(231) Left(345) Top(172) Width(614)
Define_Com Class(#PRIM_PHBN) Name(#HELLO) Caption('Hello') Displayposition(1)
Height(23) Left(387) Parent(#COM_OWNER) Tabposition(1) Top(8) Width(69)
Define_Com Class(#PRIM_PHBN) Name(#WORLD) Caption('World') Displayposition(2)
Height(23) Left(387) Parent(#COM_OWNER) Tabposition(2) Top(45) Width(69)
Define_Com Class(#PRIM_PHBN) Name(#CLEAR) Caption('Clear') Displayposition(3)
Height(23) Left(387) Parent(#COM_OWNER) Tabposition(3) Top(82) Width(69)
Define_Com Class(#STD_TEXT.Visual) Name(#STD_TEXT) Displayposition(4)
Height(19) Left(28) Marginleft(0) Parent(#COM_OWNER) Tabposition(4) Top(37)
UsepickList(False) Width(325)
```

7. Click on the  $\Box$  icon next to the first Define_Com. Your code should now look like the following:

```
Function Options(*DIRECT)
-Begin_Com Role(*EXTENDS #PRIM_FORM) Clientheight(193) Clientwidth(598)
Height(231) Left(345) Top(172) Width(614)
Define_Com Class(#PRIM_PHBN) Name(#HELLO) Caption('Hello') Displayposition(1)
Height(23) Left(387) Parent(#COM_OWNER) Tabposition(1) Top(8) Width(69)
```

The Define_Com statements have been compressed, saving space. The

*Options / Settings* dialog for *Source* can make this appearance the default.

## **Step 14. Understanding Events**

1. Drag and drop field STD_DESCL on to the form. Set up its properties as follows:

Property	Value
LabelPosition	Тор
Caption	Display the results
LabelHorAlignment	Left
LabelType	Caption
Width	374

2. Change the properties of the first field STD_TEXT as follows

Property	Value
LabelPosition	Тор
Caption	Enter some text
LabHorAlignment	Left
LabelType	Caption

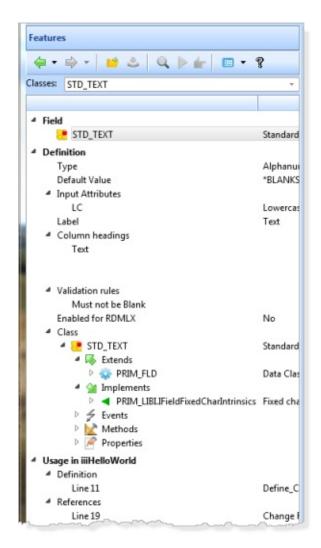
Your form should now look like the following:

	• • • • • •	
aAbBcCdDeEfFgGhHiIjJkKILmMnNoOpPqQrRsStTuUvVwWxXyY		Hello
	• • • • •	
Display the results		World
aAbBcCdDeEfFgGhHiIjJkKILmMnNoOpPqQrRsStT		Clear

- 3. Select the first field STD_TEXT and create a *Changed* event for it. Remember you can do this either using the *Events* tab on the *Details* tab, or using the right mouse menu / *Events* : *STD_TEXT*.
- 4. With the STD_TEXT selected, press F2. Alternatively use the right mouse menu and select *Field* : *STD_TEXT / Features*.

Enter some text	•	Hello	
aAbBcCdDeEfFgG	Field: STD_TEXT	🔰 📂 Open	
Display the Results aAbBcCdDeEfFgG	Delete Component	X Delete from Repository	
	Copy Component Cut Component	Quick Export	
	Add Popup Menu	Check Out Readonly	
	Goto Definition	Сору	
	Save Definition Save PRIM_EVEF Definition	Copy Name     Arrow Print	
	Events : STD_TEXT	Properties	
		Cross References	

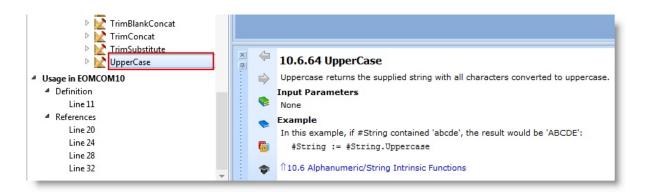
The Features tab shows the events, properties and methods for the selected component, in this case field STD_TEXT.



5. Click on the *Views* button in the tab toolbar and choose Category View:



- 6. Expand the *Intrinsics* for this type of field. Scroll down and note that there is an *UpperCase* method.
- 7. Double click on the *UpperCase* method to see the *Feature Help* for it.



8. Add the following code to the STD_TEXT.Changed event routine. New code is shown in bold.

Evtroutine Handling(#STD_TEXT.Changed) Options(*NOCLEARMESSAGE #STD_DESCL := #STD_TEXT.upperCase Endroutine

9. Change the CHANGE statement in the CLEAR.Click event to include STD_TEXT.

Change Field(#STD_TEXT #STD_DESCL) To(*blanks)

10. Recompile and test your form. Type into the first field and observe the output in results field. Your form should look like the following:

II Hello World	
Enter some text	
abc	Hello
Display the results	World
ABC	Clear

Observe that for every change event (character typed) the STD_TEXT.Changed event routine is executed and replaces STD_DESCL with the new value of STD_TEXT.

- 11. Change the STD_TEXT.Changed event routine logic to the following:
  #STD_DESCL := #STD_TEXT.upperCase.Reverse
- 12. Recompile your form. Make sure the form was closed before recompiling.

13. Test your form. Type into the first field. The results should look like the following:

Hello World	
Enter some text	
abc	Hello
Display the results	World
CBA	Clear

14. Clear the fields using the Clear button. Click on the Hello button. Your form should look like the following:

🛛 Hello World	
Enter some text	
Hello	Hello
Display the results	World
	Clear

The Changed event for field STD_TEXT has not been triggered.

Programmatic changes to a component (in this case a field) do not trigger its events.

### **Step 15. Using Component Properties**

- 1. Drag and drop a Check box to the top of your form. Note it is automatically named CKBX_1.
- 2. Set up the Check box properties as follows:

Property	Value
----------	-------

Caption Allow Uppercase and Reverse

ButtonState Checked

Your form should look like the following:



3. Change the STD_TEXT Changed event to execute the assign command, if the check box is Checked. Your code should look like the following:

Evtroutine Handling(#STD_TEXT.Changed) Options(*NOCLEARMESSAGE If (#CKBX_1.ButtonState = Checked) #STD_DESCL := #STD_TEXT.Uppercase.Reverse Endif Endroutine

Your IF expression is a Boolean expression which "GETs" the property ButtonState for component CKBX_1.

4. Compile and test your form. The results field will not be populated, if the Check box is unchecked.

- 5. In the *Design* view select the *Clear* button and change its *Enabled* property to **False**.
- 6. Make the following changes to your form logic:
  - a. Change the Hello click event to SET the Clear button to Enabled=True
  - b. Change the World click event to SET the Clear button to Enabled=True
  - c. Change the STD_TEXT field Changed event to SET the Clear button to Enabled=True
  - d. Change the Clear button click event to SET the Clear button to enabled=False

For example, the Hello Click event should look like the following:

```
Evtroutine Handling(#HELLO.Click)
#STD_TEXT := #STD_TEXT + 'Hello '
#CLEAR.enabled := true
Endroutine
```

This uses the ASSIGN statement to SET a property.

A longer form of this code could be used to set a component's property:

Set Com(#CLEAR) Enabled(true)

7. Compile and test your form. The Clear button should initially be disabled. The Clear button should be enabled whenever the Hello or World buttons are used, or when text is typed into the "Enter some text" field.

## **Step 16. Understanding Component Methods**

1. In the *Design* view, click anywhere on the form and select the *Methods* tab on the *Details* tab.

iiiHelloWorld	
Properties Events Methods ActivateForm CloseForm CloseForm HideForm MaximizeForm MinimizeForm RestoreForm SetFocus ShowForm ShowModalForm	Allow Uppetcase a Enter some text aAbBcCdDeEfFgGhHi Display the Results aAbBcCdDeEfFgGhHi

Note that your form has a *CloseForm* method.

 Your form iiiHelloWorld, inherits events, properties and methods from its Ancestor PRIM_FORM. In the *Source* tab, note the Begin_Com statement's Role(*EXTEND #PRIM_FORM)

```
Function Options(*DIRECT)
B-Begin_Com Role(*EXTENDS #PRIM_FORM) Clientheight(123) Clientwidth(467)
Height(161) Left(345) Top(172) Width(483)
Define_Com Class(#PRIM_PHBN) Name(#HELLO) Caption('Hello') Displaypositi
Usight(222) Left(2222) Devent(#COM_COMUPD) Tabuacities(1) Tabuacities(2) Night()
```

- 3. In the *Design* view, drag and drop a push button onto form.
- 4. Change the Push Button *Name* to **CLOSE**, change its *Caption* to **Close** and create a **CLOSE.Click** event. Add the following code to this Click event:

#com_owner.CloseForm

The generic name COM_OWNER can be used to refer to the current component.

Your form should look like the following:

Allow Uppercase and Reverse		
Enter some text	· · · · · · · · · · · · · · · · · · ·	
$aAbBcCdDe {\c EfFgGhHiljJkKlLmMnNoOpPqQrRsStTuUvVwWxXyY}$	· · · · · ·	Hello
Display the results	· · · · · · · · · · · · · · · · · · ·	World
aAbBcCdDeEfFgGhHiljJkKlLmMnNoOpPqQrRsStT		Clear
		Close
		:::::::::::

5. Compile and test your form. Click on the Close button to close it.

### Summary

#### **Important Observations**

- As you set component properties (names, captions, and so on, in the *Details* tab) the DEFINE_COM statements are updated in the RDML code. Alternatively, you can directly edit the DEFINE_COM statements.
- There are many different ways to perform the same operation, such as using commands in a ribbon or context menu options.
- There are many ways of coding RDML statements to achieve the same results. For example, the following two statements will produce the same result:

```
CHANGE FIELD(#STD_TEXT) TO(*BLANKS)
#STD_TEXT := *BLANKS
```

- The DEFINE_COM commands may be hidden using the editor settings.
- The F2 Feature help provides information about component properties, events and methods
- If you know LANSA functions, you need to understand there is a fundamental difference between forms and functions in the programming paradigm. Functions are procedural programs that execute in a "top to bottom" fashion. Forms use an event-driven paradigm based on graphical user interfaces. The form waits for a specific event to occur and then executes the event. Once the event has completed, control is passed back to the interface. Hence, the location of the event routines in a form's code is not important to the execution of the program.

# **Tips & Techniques**

- Review all of the editor settings to determine the options that you want to use when working in Visual LANSA.
- When editing source code, using the Auto Hide tab feature will enable the maximum amount of space for viewing your source code.
- F2 feature help provides details about components. The F1 online help will provide details about the user interface, commands, and options.
- Use the dialog box launcher in the *Compile* group in the ribbon if you need to specify the compile settings. If you submit compiles using Compile command, the compile options will simply default to the last values set in

the Compile options dialog.

The basic rules you need to remember when writing code:

- Only one command is allowed per line in the editor.
- You cannot have any leading blanks before a command.
- A command may have no parameters (as in END_COM) or may have many parameters (as in BEGIN_COM).
- Many command parameters are optional.
- Parameters may have one or more values (as in ROLE(*EXTENDS #PRIMFORM)). A blank space is used to separate values in a list.
- When coding parameters, do not leave a space between the parameter keyword and the opening bracket. For example, OPTIONS(*DIRECT) or OPTIONS( *DIRECT ) are correct, but OPTIONS (*DIRECT) is not correct.
- When parameters use their default values, the parameters are not explicitly shown in the command, but they can be viewed in the *Command Assistant* tab. Select the line of code and use F4 to display the *Command Assistant* tab. It is often convenient to float and resize this tab so that the command parameters can be seen more easily.
- An asterisk * in the first position of a line is used for a comment. You can use the short cut keys CTRL+W and CTRL+Shift+W to comment or uncomment a single line or selected block of lines.

# What You Should Know

- The basic structure of the code in a form
  - FUNCTION
  - BEGIN_COM
  - DEFINE_COM
  - EVTROUTINE
  - ENDROUTINE
  - END_COM
- How the form and components are related to the source.
- How to create a form.
- How to add components to a form.
- How to change the properties of a component on a form using the Details

tab.

- How to define events for a component.
- How to compile and execute a form.
- The basic structure of LANSA commands.
- How to display feature help.
- How to find intrinsic methods.
- How to use intrinsic methods.
- How to use the RDML/RDMLX commands.
  - CHANGE
  - ASSIGN
- How to write a Boolean expression.

# FRM025 - Insert a Database Record

#### **Objectives:**

• To create an Add Employee form.

Add Employee		
Employee Number	A2000	
Employee Surname	Franklin	
Employee Given Name(s)	Mary	
Street No and Name	58 Surrey St	
Suburb or Town	Sydney	
State and Country	NSW	
Post / Zip Code	2010	
Home Phone Number	956800045	
Business Phone Number	099485494	
Department Code	ADM	
Section Code	01	
Employee Salary	80,000.00 ‡	Save
Start Date (DDMMYY)	111111	Save
		Close
Start date is not valid - press l	Help function key	

- To introduce the INSERT, GROUP_BY and MESSAGE commands
- To introduce the loop commands
  - BEGIN_LOOP/END_LOOP
  - DOWHILE/ENDWHILE
  - DOUNTIL/ENDUNTIL
- To understand how field and file level validation is handled in a form
- To use the Status Bar component to display messages
- To use RDMLX style string handling.
- To implement "busy cursor"
- To review other "delay" feedback techniques

To achieve these objectives you must complete the following:

Step 1. Create form iiiAddEmploy – Add Employee

Step 2. Add Fields to the Form

Step 3. Add Push Buttons and Click Event Logic

Step 4. Using a Busy Cursor

Summary

# **Before You Begin**

You must have completed:

FRM015 - Getting Started with Forms Programming

## Step 1. Create form iiiAddEmploy – Add Employee

1. On the *File* menu, use the *New* button to create a *Basic Form* defined as follows:

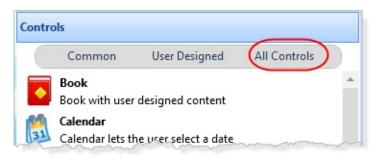
Name iiiAddEmploy

Description Add Employee

Enabled for RDMLX? 🗹 Yes

**Note:** When creating a form you should select a specific Framework, such as *Personnel & Payroll*. Frameworks allow you to organize components (forms, reusable parts and WAMs) into logical groups. Your own Visual LANSA development work should use Frameworks which you define at the LANSA partition level on the IBM i server. All components have a property ComponentFramework which may be changed if required.

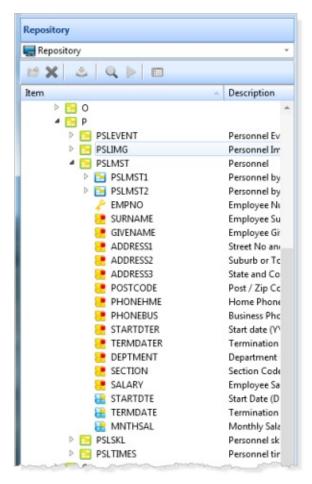
2. In the *Design* view, select the *Controls* tab and select *All Controls*:



3. Drag and drop a *Status Bar* component onto the form. It will be attached to the bottom of the form.

## Step 2. Add Fields to the Form

- 1. Select the *Repository* tab. Expand the *Files* group. If necessary change the Files group to *Alphabetic Grouping* by using the right mouse menu on the Files group.
- 2. Expand the P group and expand the file PSLMST. Your Repository tab should look like the following:



- 3. Enlarge the form by dragging its lower edge.
- 4. On the *Repository* tab, hold down the *Shift* key and use the left mouse button to select EMPNO and then PHONEBUS. All fields between these will be selected. This is a standard Windows *select a group of objects* technique.
- 5. Drag these fields onto the form, towards the top.

	•	G	-
🐸 🗙 😃	Q 🕨 🗉	E	
*	Description		
þ	*		
PSLEVENT	Personnel Event Log		
B PSLIMG	Personnel Images		
PSLMST	Personnel		
PSLMST1	Personnel by Deptment, Sect		
PSLMST2	Personnel by Surname, Giver		
EMPNO	Employee Number		
C SURNAME	Employee Surname		
GIVENAME	Employee Given Name(s)		
ADDRESS1	Street No and Name		
ADDRESS2	Suburb or Town		
ADDRESS3	State and Country		
POSTCODE	Post / Zip Code		
PHONEHME	Home Phone Number		
PHONEBUS	Business Phone Number		
CTARTOTO	0 · · · · · · · · · · · · · · · · · · ·		

6. Select the group of fields from DEPTMENT to STARTDTE and drag these onto the form just below the others.

**Note:** STARTDTE is a virtual field which updates the real field STARTDTER.

7. Click on the form to unselect the last group of fields. With the Shift key held down, select EMPNO and then select each of the other fields. Then use the *Align* dialog from the *Design* tab, to align the fields on the left, spaced equally.

Your form should look like the following:

ABCDE
aAbBcCdDeEfFgGhHiljJ
aAbBcCdDeEfFgGhHiljJ
aAbBcCdDeEfFgGhHiljJkKILm
aAbBcCdDeEfFgGhHiljJkKlLm
aAbBcCdDeEfFgGhHiljJkKlLm
123456
ABCDEFGHUKLMNO
ABCDEFGHUKLMNO
ABCD
AB
123,456,789.12
123456

8. Save your form.

#### Step 3. Add Push Buttons and Click Event Logic

- 1. On the *Controls* tab, select the *Common* group of controls. Drag and drop two *Push Buttons* onto the bottom right hand side of the form.
- 2. Use the *Details / Properties* tab to set up the two push buttons as follows:

Property	Value
Caption	Save
Name	SAVE

Property	Value
Caption	Close
Name	CLOSE

Your form should look like the following:

Employee Number	ABCDE		
Employee Surname	aAbBcCdDeEfFgGhHiljJ		
Employee Given Name(s)	aAbBcCdDeEfFgGhHiījJ		
Street No and Name	aAbBcCdDeEfFgGhHiljJkKILm		
Suburb or Town	aAbBcCdDeEfFgGhHiljJkKILm		
State and Country	aAbBcCdDeEfFgGhHiljJkKILm		
Post / Zip Code	123456		
Home Phone Number	ABCDEFGHIJKLMNO		
Business Phone Number	ABCDEFGHUKLMNO		
Department Code. Section Code	ABCD		
Employee Salary	123,456,789.12 Sav	e	
Start Date (DDMMYY)	123456 Clo		

3. Use the *Details / Events* tab to create a Click event for each button.

4. Add code to the CLOSE Click event to close the form. Your code should look like the following:

Evtroutine Handling(#CLOSE.Click) #com_owner.closeForm Endroutine

5. Define a GROUP_BY to include all fields on the form. Your code should look like the following

Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME #A

Notes:

- Group_by statements are usually defined at the top of the program, following the component definition. This makes these declarations easy to find, when maintaining a program.
- You should always use the *Command Assistant (F4)* to complete this type of statement, where a set of field names is required.
- Use the *Editor Options* and use the *General* settings to set *Assistant* to *Auto Expand Parameters*.
- 6. Type GROUP_BY and press *F4*.
  - a. In the *Command Assistant*, enter the name of the Group_by **#EMPDATA.**
  - b. Position the cursor in the *Fields* parameter.

**Note:** You may prefer to float and resize the Command Assistant tab so that it easily shows more information.

NAME	#EMPDATA	$\checkmark$
- FIELDS Fields and Attributes		Commands Variables Fields by F
Fields and Attributes		New qualification #ADDRESS1
		Name
		ADDRESS1
		ADDRESS3
		📜 📜 DEPTMENT
		EMPNO

7. Select the *Fields by File* tab and enter PSL in the *Filter / File Name*:

NAME - FIELDS Fields and Attributes	#EMPDATA	Commands Variables Result value filters – File name File des	· · · · · · · · · · · · · · · · · · ·	
		psl	cription	Library na *first
		Item	<ul> <li>Description</li> </ul>	Details
		<ul> <li>PSLEVENT</li> <li>PSLEVENTA</li> <li>PSLEVENTB</li> </ul>	Personnel Event Log PSLEVENT by event da PSLEVENT by type, dat	

8. Expand the file PSLMST and hold down the *Shift* key, select EMPNO and then PHONEBUS and press Enter. Your Assistant dialog should now look like the following:

NAME	#EMPDATA	$\checkmark$		
- FIELDS	#EMPNO #SURNAME #GIVEN			
Fields and Attributes	#EMPNO	Commands Variables Fields by File	Repository Fields Groups and Lists Special Value	
Fields and Attributes	#SURNAME	New qualification #ADDRESS1		
Fields and Attributes	#GIVENAME	Name	<ul> <li>Description</li> </ul>	
Fields and Attributes	#ADDRESS1	ADDRESS1	Street No and Name	
Fields and Attributes	#ADDRESS2	ADDRESS2	Suburb or Town	
Fields and Attributes	#ADDRESS3	ADDRESS		
Fields and Attributes	#POSTCODE		State and Country	
Fields and Attributes	#PHONEHME	DEPTMENT Department Code		
Fields and Attributes		EMPNO Employee Number		
		GIVENAME	Employee Given Name(s)	
		PHONEBUS	Business Phone Number	
		PHONEHME Home Phone Number		

- 9. Position the cursor in the empty *Fields and Attributes* entry below PHONEBUS.
- 10. Select *Fields by File* again and select the group of field from DEPTMENT to STARTDTE and press enter. Your *Command Assistant* dialog should now look like the following:

NAME	#EMPDATA	-	$\checkmark$			
- FIELDS	#EMPNO #SURNAME #GIVE					
Fields and Attributes	#EMPNO		Commands		Fields by File	Repos
Fields and Attributes	#SURNAME		New qualifica	ation	#DEPTMENT	
Fields and Attributes	#GIVENAME		Name			
Fields and Attributes	#ADDRESS1		E ADDR	2ESS1		
Fields and Attributes	#ADDRESS2	Ξ	ADDF			
Fields and Attributes	#ADDRESS3					
Fields and Attributes	#POSTCODE			MENT		
Fields and Attributes	#PHONEHME					
Fields and Attributes	#PHONEBUS		EMPI			
Fields and Attributes	#DEPTMENT			NAME		
Fields and Attributes	#SECTION		е рног	NEBUS		
Fields and Attributes	#SALARY	-	4			

11. Click on the **tick** highlighted above and close the *Command Assistant*. Your Group_by should look like the following:

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

12. Add an INSERT command to the SAVE.Click event routine. This should insert the Group_by EMPDATA to the file PSLMST. Your event routines code should look like the following:

Evtroutine Handling(#SAVE.Click) Insert Fields(#EMPDATA) To_File(PSLMST) Endroutine

- 13. Compile your form and execute it.
- 14. Without entering any data, click the *Save* button; Your form should look like the following:

Add Employee			
Employee Number			
Employee Surname			
Employee Given Name(s)			
Street No and Name			
Suburb or Town			
State and Country			
Post / Zip Code	0		
Home Phone Number			
Business Phone Number			
Department Code			
Section Code			
Employee Salary	.00	<b>\$</b>	<b>6</b>
Start Date (DDMMYY)	000000		Save
			Close
naloves number not in rar	as "A0000" to "A0000"		
mployee number not in rar	ge A0000 to A9999		

15. Click on the 🔽 down icon on the *Status Bar* to see all the validation error messages.

Having completed the *Fundamentals Repository* module you should know that the error messages have been generated by the file OAM.

Error messages are automatically routed to the *Status Bar*. Fields with validation errors are automatically highlighted.

16. The INSERT command has a default parameter VAL_ERROR(*LASTDISPLAY). In a form this will branch to the ENDROUTINE for the event or method routine. To demonstrate this point, add the following MESSAGE command after the INSERT command in the SAVE.Click event routine.

Message Msgtxt('This message only displayed on successful INSERT')

Recompile your form and retest "add a blank record". Scroll to the end of the status bar messages. The above message will not be shown.

In your applications, you will often change the validation error parameter on I/O commands to VAL_ERROR(*Next), so that your own logic can handle error conditions.

17. Insert valid data to add a new employee. Employee numbers in the series starting A2000 should be available.

**Note:** Post/Zip code must be in the range 2000 to 7999 (Australian). Department codes ADM, FLT, GRP, MKT and SLS should exist with Sections codes of 01, 02 and 03.

Observe that your "This message displayed on successful INSERT" is now displayed.

Note also that at this stage the fields have not been re-initialized.

18. Change the message following the INSERT command as follows:

Message Msgtxt('Employee number ' + #EMPNO + ' has been added')

Note that RDMLX enables this style of code to be written.

In your own applications you will usually use a message file message for this type of feedback.

19. After the INSERT, add code to reset the employee fields to their default values, using the *Group_by*:

#EMPDATA := *default

20. Recompile your form and retest it by adding another employee record. All field values should be reset to their default values as defined in their *Repository* field definitions, in this case they are all blank or zeroes.

Note that your new message **Employee number A9999 has been added** is displayed in the status bar.

21. With your form still running after the successful insert, try to insert another blank record. Review the status bar messages and note that the "Employee

number A9999 has been added" has been cleared.

An event routine has an OPTIONS setting with default values as follows: Evtroutine Handling(#SAVE.Click) options(*CLEARERRORS *CLEARMES

Each time an event routine is executed, by default, it clears field errors and messages.

## Step 4. Using a Busy Cursor

Your applications should provide good feedback to the user, for example when routine will take more than a few seconds to complete. This step illustrates how to implement a *Busy Cursor*. Of course, in reality you wouldn't need this for a routine which performs a single record add.

 Event handling routines have a Com_Cursor() parameter which may have values of: DEFAULT, *DELAY_01, *DELAY_02, *DELAY_04, *IMMEDIATE or *NEVER

The DELAY values are seconds.

- 2. Change the SAVE.Click routine to have Com_Cursor(*IMMEDIATE).
- 3. Use a Begin_Loop/End_Loop to add a delay at the start of the SAVE.Click routine. For example, your routine could look like the following:

Evtroutine Handling(#SAVE.Click) Com_Cursor(*IMMEDIATE)

#std_num := 0

Begin_Loop Using(#std_num) To(500000)

```
#std_num += 1
```

End_Loop

Insert Fields(#empdata) To_File(pslmst)

```
Message Msgtxt('Employee number ' + #EMPNO + ' has been added')
#empdata := *default
Endroutine
```

**Note:** You may need to use **To(10,000,000)** for the Begin_Loop if you have a fast PC, in order to produce an noticeable delay.

Alternatively, you could have added a loop structure using DOWHILE/ENDWHILE or DOUNTIL/ENDUNTIL.

4. Recompile your form and test it, for example by trying to add a blank record. You should see the busy cursor displayed for a few seconds every time:

Employee Number		
Employee Sumame		
Employee Given Name(s)		
Street No and Name		
Suburb or Town		
State and Country		
Post / Zip Code	0	
Home Phone Number		
Business Phone Number		
Department Code		
Section Code		
Employee Salary	.00	
Start Date (DDMMYY)	000000	

Other "delay" feedback techniques available include a *Progress Bar* component and showing "stop" and "go" images.

### Summary

#### **Important Observations**

- GROUP_BY's define a set of fields and simplify your code and reduce future maintenance effort
- The MESSAGE command can display text messages or display a message from a message file
- Database command have a VAL_ERROR() parameter, with a default value of *LASTDISPLAY. On error, this setting will branch to the routines EndRoutine statement.
- Event handling routines have a Com_Cursor parameter, which enables "busy cursor" to be implemented.

# **Tips & Techniques**

- Always use the Command Assistant with commands such as GROUP_BY which require a list of field names
- Always consider whether to provide the user with additional feedback such as "busy cursor".

# What I Should Know

- How to define a GROUP_BY command
- How to define a MESSAGE command
- How to use the INSERT command
- How to define a BEGIN_LOOP/END_LOOP command
- How to use the Status Bar component to display application messages
- How to implement "busy cursor"
- Be aware of other forms of "delay" feedback are available.

# FRM035 - Maintain a Simple Database Table

### **Objectives:**

- To introduce the most common forms of database access including the following commands:
  - FETCH
  - INSERT
  - UPDATE
  - DELETE.
- To learn about error handling using the IF_STATUS command when using file operations.
- To learn how to use some of the program level validations commands:
  - BEGINCHECK / ENDCHECK
  - CALLCHECK
  - CONDCHECK
  - DATECHECK
  - FILECHECK
  - RANGECHECK
  - VALUECHECK
  - IF_ERROR, SET_ERROR.
- To understand how program validations relate to the repository validation performed by the Object Access Modules.
- To understand how to use a Built-In Function (BIF) to display a message box with user confirmation.
- To create a simple application to maintain the DEPTAB file.

Department Maintena	nce	
Department Code	ADM	Fetch
Department Description	ADMINISTRATOR DEPT	Insert
Standard Number	0	Update
	Clear	Delete

To achieve these objectives, you need to complete the following:

- Step 1. Create a Department Maintenance Form
- Step 2. Fetch Existing Data from a File
- Step 3. Insert Data to a File
- Step 4. Add Program Level Validations
- Step 5. Update Data in a File
- Step 6. Delete Data from the File
- Step 7. Update and Delete Last Record Read

Summary

## **Before You Begin:**

In order to complete this exercise, you should have completed the previous exercise.

You may wish to review the following topics in the *Technical Reference Guide*:

• RDML Commands

and

• RDMLX Commands and RDMLX Features.

# Step 1. Create a Department Maintenance Form

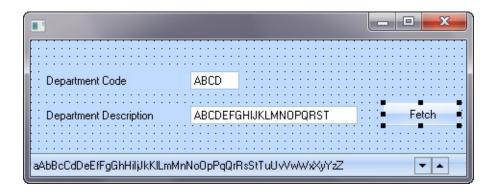
In this step you will create a simple form to fetch a record from the Department file.

In the later steps, you will add insert, update and delete operations to this form to build a complete maintenance application.

- 1. Create a new, basic form named iiiMntDept *Database Access* (where iii are your course assigned initials). If you are using iii=DEM, your component must be named DEMCOM03.
- 2. Using the *Repository* tab, locate the DEPTAB file. Notice that the file has two fields DEPTMENT and DEPTDESC. The file is keyed on DEPTMENT.

Repository				
Repository 🔹				
🔓 New 🔻 🔛 🗶 💩	Q 🕨 🗉			
Item 🔺	Description			
ශ් 🔁 DC@W11	L4W3 - Banner Details 🛛 🔺			
DC@W12	L4W3 - LANSA system table			
DC@W14	Web Event Links			
DC@X01	XML Page Header			
DC@X02	XML Page Detail			
DC@X03	XML Component Registry			
🖻 🔁 DC@XAA	XML Application			
🖻 🔁 DC@XAP	XML Application Property			
🖻 🔁 DC@XPR	XML Property			
DC@XPV	XML Property Value			
🗖 🔁 DEPTAB	Department code table			
🔑 DEPTMENT	Department Code			
📜 DEPTDESC	Department Description			

- 3. Drag and drop the DEPTMENT and DEPTDESC fields on to the form.
- 4. Add a *Status Bar* to the form so that validation error messages can be displayed.
- 5. Add a *Push Button* to the form.
  - a. Set the button *Name* and *Caption* to Fetch.
  - b. Create a *Click* event routine for the Fetch button.
- 6. Your form should appear like this:



## Step 2. Fetch Existing Data from a File

In this step you will use the FETCH command to retrieve a single record from the database. You should notice that there are no OPEN or CLOSE statements required for the file. File opening and closing is handled automatically by LANSA.

1. In the *FETCH.Click* event routine, add a FETCH command to retrieve the DEPTMENT and DEPTDESC fields from the DEPTAB file. For this first example, you want to FETCH the record where DEPTMENT='ADM'.

Your code should appear as follows:

EVTROUTINE HANDLING(#FETCH.Click) **FETCH FIELDS(#DEPTMENT #DEPTDESC) FROM_FILE(DEPTAB)** ENDROUTINE

**Reminder:** You can use F1 to display the online help for more details about commands.

- 2. Compile and execute the form.
  - a. Press the *Fetch* button.

The record for department ADM is read from the database and the result displayed on the form. If for some reason this record does not exist, a message is automatically issued due to the ISSUE_MSG(*YES) parameter on the FETCH command . In your own applications you will usually want to remove this option and add your own error handling.

Department Maintenance		_ <b>D</b> _ X
Department Code Department Description	ADM	Fetch

3. Typically, you want to allow the user to enter a Department Code for the record to be read. The value that the user has entered for the DEPTMENT field on the form will be used to fetch the record.

If the record is not found, appropriate error messages must be displayed to

the user. Review the use of the IF_STATUS command to check that the FETCH was successful.

I/O Command Return Codes Table

Command	I/O Error	Dictionary Validation	Not Found	Found Or Completed
INSERT	ER	VE*	-	OK
UPDATE	ER	VE	NR	OK
FETCH	ER		NR	OK
SELECT	ER		EF#	OK
WHERE				OK
FILECHECK	-		NE	EQ
CHECK_FOR	-		NE	EQ
DELETE	ER	VE	NR	OK

* An attempted INSERT with a duplicate key will return VE.

# A SELECT command using a WHERE parameter will select each record and test for the condition. When the last record is selected, the processing will leave the SELECT loop with the data from the last record selected. This record may not have met the WHERE condition.

Your finished code should appear as follows:

EVTROUTINE HANDLING(#FETCH.Click) FETCH FIELDS(#DEPTMENT #DEPTDESC) FROM_FILE(DEPTAB) \ IF_STATUS IS_NOT(*OKAY) MESSAGE MSGTXT('Error retrieving Department') ENDIF ENDROUTINE

- 4. Compile and execute the form.
  - a. Leave the Department Code blank and press the *Fetch* button.

The error message will be displayed.

Department Maintenance	
Department Code	Fetch
Error retrieving department	

b. Enter a Department Code of ADM and press the *Fetch* button.

The ADM record is read from the database and the result displayed on the form.

Department Maintenance		
Department Code Department Description	ADM	Fetch

5. Close the form.

# Step 3. Insert Data to a File

In this step you will let the user insert new data into the DEPTAB file.

You will add an *Insert* push button to the form along with the STD_NUM field. The STD_NUM field is not used in the DEPTAB file but will used to help demonstrate program level validation rules. You will also use a GROUP_BY statement to simplify the code in your form.

- 1. Drag and drop the STD_NUM field to the form.
- 2. Drag and drop a push button to the form.
  - a. Set the button *Name* and *Caption* to INSERT.
  - b. Create a *Click* event routine for the INSERT button.

**Note:** The controls on a form (fields, push buttons etc) have a *TabPosition* property. You as you add controls to the form you should ensure their *TabPosition* is set to an appropriate value. For example the three fields should be *TabPosition* = **1**,**2** and **3** 

3. Your form should appear like this:

Department Code	ABCD	
Department Description	ABCDEFGHIJKLMNOPQRST	Fetch
Standard Number	1,234,567	Insert (
aAbBcCdDeEfFgGhHiljJkKlLmM	nNoOpPqQrRsStTuUvVwWxXyYzZ	•

4. For each of the database commands, you will be including a list of fields to fetch, insert, update and delete. To simplify your I/O statements, add a GROUP_BY command after your DEFINE_COM statements so that you can refer to all the fields by the group name:

GROUP_BY NAME(#FORMDATA) FIELDS(#DEPTMENT #DEPTDESC #STD_NUM)

Once added, you change the FETCH command as follows: FETCH FIELDS(#FORMDATA) FROM_FILE(DEPTAB)

### WITH_KEY(#DEPTMENT)

Notice that the STD_NUM field can be included in the GROUP_BY used by the FETCH even though it is not a field in the DEPTAB file. In database operations the STD_NUM field will be ignored, but used in other operations where the group is used.

5. In the INSERT.Click event routine, add an INSERT command to add a new record to the file.

Remember to add the appropriate error checking. Once the INSERT completes successfully, all fields on the form should be reset to their repository defaults.

Your finished code should appear as follows:

```
EVTROUTINE HANDLING(#INSERT.Click)
INSERT FIELDS(#FORMDATA) TO_FILE(DEPTAB)
IF_STATUS IS(*OKAY)
MESSAGE MSGTXT('Department inserted successfully')
#FORMDATA := *DEFAULT
ELSE
IF_STATUS IS(*ERROR)
MESSAGE MSGTXT('Error inserting Department')
ENDIF
ENDIF
ENDIF
```

Note that the INSERT command has many parameters which are not shown in the editor when their default values are used. The INSERT command you have created looks like this with default values shown:

```
INSERT FIELDS(#FORMDATA) TO_FILE(DEPTAB) IO_STATUS(*STATU
VAL_ERROR(*LASTDIS) ISSUE_MSG(*NO) RETURN_RRN(*NONE) CH
AUTOCOMMIT(*FILEDEF)
```

- 6. Compile and execute the form.
  - a. Leave the Department Code and Description blank and press the *Insert* button.

Notice that fields in error are highlighted and error messages are

#### displayed.

	Database Maintenance	_ 🗆 🗙
Department Code Department Description Standard Number	0	Fetch
Department code is required	1	

- b. Review the error messages. They are caused by the validation rules in the Repository. When the INSERT fails due to these repository errors, the program automatically returns to the last display. This is controlled by the default VAL_ERROR(*LASTDIS) parameter on the INSERT command.
- c. Enter a Department Code of ADM and press the *Fetch* button to retrieve an existing record.
- d. Press the *Insert* button to try to duplicate this data in the database.

Notice that no fields are highlighted in error as the repository validation rules have been satisfied. The error message is automatically generated by LANSA.

	Database Maintenance	- • ×
Department Code	ADM	Fetch
Department Description	Administration	Insert
Standard Number	0	iniser()

e. Enter a Department Code of **III** (where iii = your initials) and enter your name for the Department Description. Enter the number 999 into Standard Number. Press the *Insert* button.

(Remember STD_NUM is not used in the DEPTAB file.)

The new record should be inserted into the database, a message displayed, and all fields reset to their default.

7. Close the form.

## Step 4. Add Program Level Validations

In this step you will learn how to create program level validations using code within a BEGINCHECK/ENDCHECK block.

Note that this exercise is just an example created to explain validation commands. Most of your validation rules will be placed in the *Repository* as part of the file definitions. You may also design your forms to reduce the number and types of program validations required. For instance, a drop down list box provides a list of values so that VALUECHECK is not required.

1. Add a validation check that ensures the Department Code does not contain any embedded blanks.

For example, 'A A' should not be allowed as a valid code. Use a CONDCHECK statement within a BEGINCHECK / ENDCHECK block and the CONTAINS intrinsic field method to search for blank characters.

Add the following code immediately before the INSERT command in the *INSERT.Click* event routine:

BEGINCHECK CONDCHECK FIELD(#DEPTMENT) COND(#DEPTMENT.Contains( ' ' )) I IF_FALSE(*NEXT) MSGTXT('Code cannot contain embedded blanks.') ENDCHECK

Note: CONDCHECK. . . . . IF_FALSE( . . . . is a single command, on a single line. It is shown here on two lines due to space limitations. You should type CONDCHECK and then use the *Command Assistant (F4)* to complete this code.

- 2. Compile and execute the form.
  - a. Enter a Department Code that includes a blank space and press the *Insert* button.

Notice the field in error and the error message.

5	Database Maintenance	_ 🗆 ×
Department Code	AA	Fetch
Department Description	Department A	Insert
Standard Number	0	

The repository validation rules (checking that Department Description is not blank) have not been invoked because the INSERT command has not yet been executed because of the error detected in BEGINCHECK/ENDCHECK.

- 3. Close the form.
- 4. Another option to using the CONDCHECK is to use a simple IF statement combined with the SET_ERROR command. Using SET_ERROR allows more than one field to be set in error.
  - a. Add a rule that checks that the user has not entered the same values for the DEPTMENT and DEPTDESC fields. If they are the same, set both fields in error.

#### BEGINCHECK CONDCHECK FIELD(#DEPTMENT) COND(#DEPTMENT.Contains( ' ' )) I IF_FALSE(*NEXT) MSGTXT('Code cannot contain embedded blanks.')

#### IF COND(#DEPTMENT *EQ #DEPTDESC) SET_ERROR FOR_FIELD(#DEPTMENT #DEPTDESC) MSGTXT('Departmer as Department Description.') ENDIF

#### ENDCHECK

5. Add a RANGECHECK validation to check if STD_NUM is between 1 and 10.

Remember this field is not used in the DEPTAB file. You are simply using program level validations to check the values of the fields on the screen.

RANGECHECK FIELD(#STD_NUM) RANGE((1 10)) MSGTXT('Must be in range 1 to 10.')

6. Add a VALUECHECK validation to check if the DEPTDESC field is in a list of reserved values NONE, END or LAST.

VALUECHECK FIELD(#DEPTDESC) WITH_LIST(NONE END LAST) IN_ NOT_INLIST(*NEXT) MSGTXT('This description is reserved.')

7. Finally, check that the Department Code does not already exist in the file DEPTAB. Note that this check is not necessary as duplicate key fields are automatically checked based on the file attributes. This is just an example. Typically, you might check that the field is present in a different file.

FILECHECK FIELD(#DEPTMENT) USING_FILE(DEPTAB) USING_KEY( NOT_FOUND(*NEXT) MSGTXT('Department Code already exists.')

Your finished validation code should appear as follows:

BEGINCHECK

FILECHECK FIELD(#DEPTMENT) USING_FILE(DEPTAB) USING_KEY( NOT_FOUND(*NEXT) MSGTXT('Department Code already exists.') VALUECHECK FIELD(#DEPTDESC) WITH_LIST(NONE END LAST) IN_ NOT_INLIST(*NEXT) MSGTXT('This description is reserved.') RANGECHECK FIELD(#STD_NUM) RANGE((1 10)) MSGTXT('Must be ir CONDCHECK FIELD(#DEPTMENT) COND(#DEPTMENT.Contains( ' ' )) I IF_FALSE(*NEXT) MSGTXT('Code cannot contain embedded blanks.') IF COND(#DEPTMENT *EQ #DEPTDESC) SET_ERROR FOR_FIELD(#DEPTMENT #DEPTDESC) MSGTXT('Departm ENDIF ENDCHECK

- 8. Compile and execute the form.
  - a. Enter a Department Code of ADM and a Description of XYZ. Leave the STD_NUM as 0 and press the *Insert* button.
  - b. Notice the fields in error and scroll through the error messages displayed.

	Database Maintenance	×
Department Code	ADM	Fetch
Department Description	хуг	Insert
Standard Number	0	intoolty

c. Try entering identical values for the Department Code and Description.

9. Close the form.

It is recommended that you review information about other validation commands. Using the *LANSA Technical Reference*, you should review the CALLCHECK and DATECHECK commands.

# Step 5. Update Data in a File

In this step you will add an Update button to your form in order to update an existing record in the file.

In this first example, you will update using the key to the file. In *Step 7. Update and Delete Last* 

*Record*, you will modify the form to update the record that was fetched.

- 1. Drag and drop a push button to the form.
  - a. Set the button *Name* and *Caption* to UPDATE.
  - b. Create a *Click* event routine for the UPDATE button.
- 2. Your form should appear like this:

		- 🗆 🗡
Department Code	ABCD	Update
Department Description	aAbBcCdDeEfFgGhHiljJ	Update
Standard Number	1,234,567	- Undata
		opuare
aAbBcCdDeEfFgGhHiljJkKlLml	MnNoOpPqQrRsStTuUvVwWxXyYz	Ζ

3. In the *UPDATE.Click* event routine, add an UPDATE command to update an existing record in the DEPTAB file. Remember to add the appropriate status error checking. Once the UPDATE completes successfully, all fields on the form should be reset to their repository defaults.

Your finished code should appear as follows:

```
EVTROUTINE HANDLING(#UPDATE.Click)
UPDATE FIELDS(#FORMDATA) IN_FILE(DEPTAB) WITH_KEY(#DE
IF_STATUS IS(*OKAY)
MESSAGE MSGTXT('Department updated successfully')
#FORMDATA := *DEFAULT
ELSE
IF_STATUS IS(*NORECORD)
MESSAGE MSGTXT('Department not found')
ELSE
IF_STATUS IS(*ERROR)
MESSAGE MSGTXT('Error updating Department')
```



- 4. Compile and execute the form.
  - a. Fetch your III test record that you inserted in *Step 3*. *Insert Data to a File*.
  - b. Change the Description to XYZ and press the *Update* button.

Department Maintena	nce	
Department Code		
Department Description		Fetch
Standard Number	0	Insert
Department updated success	fully	

Notice that the program validations you added on the DEPTDESC field in the previous step are only applied when a new record is inserted with the INSERT command, but not when the record is changed with the UPDATE command. If you wanted the rules to be applied to both INSERT and UPDATE, the best solution would be to place this rule in the *Repository*.

- 5. Close the form.
- 6. Using the IF_STATUS command is the recommended technique for checking the status of file operations. An alternative technique is to use the IO\$STS field or another field to store the status code. For example:

```
DEFINE FIELD(#RETCODE) TYPE(*CHAR) LENGTH(2)
UPDATE FIELDS(#FORMDATA) IN_FILE(DEPTAB) IO_STATUS(#RETC(
```

You can now check the value of RETCODE to determine the status returned by the update. You could use a CASE statement as follows:

```
CASE OF_FIELD(#RETCODE)
WHEN VALUE_IS(= OK)
MESSAGE MSGTXT('Department updated successfully')
WHEN VALUE_IS(= NR)
```

MESSAGE MSGTXT('Department not found') WHEN VALUE_IS(= ER) MESSAGE MSGTXT('Error updating Department') WHEN VALUE_IS(= EF) MESSAGE MSGTXT('End of file.') WHEN VALUE_IS(= BF) MESSAGE MSGTXT('Beginning of file.') WHEN VALUE_IS(= EQ) MESSAGE MSGTXT('Equal key found.') WHEN VALUE_IS(= NE) MESSAGE MSGTXT('No equal key found.') OTHERWISE MESSAGE MSGTXT('Unidentified file operation return code.') ENDCASE

Note that if you do not explicitly specify the IO_STATUS option, the return code is automatically stored in the IO\$STS field. You can use a CASE statement with this field instead:

```
CASE OF_FIELD(#IO$STS)
WHEN VALUE_IS(= OK)
...
```

ENDCASE

### Step 6. Delete Data from the File

In this step you will add a Delete button to your form in order to delete an existing record in the file.

- 1. Drag and drop a push button to the form.
  - a. Set the button *Name* and *Caption* to DELETE.
  - b. Create a *Click* event routine for the Delete button.
- 2. Your form should appear like this:

		- • ×
Department Code	ABCD	Fetch
Department Description	ABCDEFGHIJKLMNOPQRST	Insert
Standard Number	1,234,567	Update
		Delete
aAbBcCdDeEfFgGhHiljJkKILmM	nNoOpPqQrRsStTuUvVwWxXyYzZ	<b>T</b>

3. In the *Delete.Click* event routine, add a DELETE command to delete a record from the file. Remember to add the appropriate status error checking. Once the DELETE completes, all fields on the form should be reset to their repository defaults.

Your finished code should appear as follows:

```
EVTROUTINE HANDLING(#DELETE.Click)
DELETE FROM_FILE(DEPTAB) WITH_KEY(#DEPTMENT)
IF_STATUS IS(*OKAY)
MESSAGE MSGTXT('Department deleted successfully')
#FORMDATA := *DEFAULT
ELSE
IF_STATUS IS(*NORECORD)
MESSAGE MSGTXT('Department not found')
ELSE
IF_STATUS IS(*ERROR)
MESSAGE MSGTXT('Error deleting Department')
ENDIF
ENDIF
```

#### ENDIF ENDROUTINE

- 4. Compile and execute the form.
  - a. Enter a Department Code of III (your test record) and press the *Delete* button.

	Database Maintenance	- 🗆 🗙
Department Code		Fetch
Department Description		Insert
Standard Number	0	
		Update
		Delete
Department deleted succes	sfully	.4

5. In many applications, users are prompted to confirm that the record is to be deleted. You can add this message using the LANSA Built-In Function MESSAGE_BOX_SHOW.

a. To accept the user response, you need to define a new field as follows:

DEFINE FIELD(#ANSWER) TYPE(*CHAR) LENGTH(6)

b. In the *DELETE.Click* event routine, invoke the MESSAGE_BOX_SHOW Built-In Function by adding the following USE command to the very start of the routine:

```
USE BUILTIN(MESSAGE_BOX_SHOW) WITH_ARGS(YESNOCANCEL ]
'Are you sure you want to delete?') TO_GET(#ANSWER)
```

c. Immediately after the USE command, check the value of #ANSWER. If the answer is YES, proceed to delete the record.

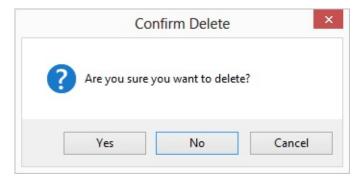
Your finished code should appear as follows:

EVTROUTINE HANDLING(#DELETE.Click) USE BUILTIN(MESSAGE_BOX_SHOW) WITH_ARGS(YESNOCANCI 'Are your sure you want to delete?') TO_GET(#ANSWER) IF COND(#ANSWER = YES) DELETE FROM_FILE(DEPTAB) WITH_KEY(#DEPTMENT) IF_STATUS 1 MESSAGE MSGTXT('Department deleted successfully') #FORMDATA := *DEFAULT ELSE IF_STATUS IS(*NORECORD) MESSAGE MSGTXT('Department not found') ELSE IF_STATUS IS(*ERROR) MESSAGE MSGTXT('Error deleting Department') ENDIF ENDIF ENDIF ENDIF ENDIF ENDIF

You should review the documentation for the MESSAGE_BOX_SHOW Built in Function in the *Technical Reference Guide*. It can also be used with MESSAGE_BOX_ADD and MESSAGE_BOX_APPEND Built in Functions to output a message box containing more text.

- 6. Compile and execute the form.
  - a. Insert some new test data.
  - b. Fetch your test record and press the *Delete* button.

A confirmation message is displayed.



7. Close the form.

# Step 7. Update and Delete Last Record Read

In this step you will learn about updating and deleting based on the last record fetched instead of using the WITH_KEY parameter.

In order to use the last record read, it is important that users cannot change the key once a record has been fetched. It is equally important that a FETCH has been performed before the update or delete is requested. In order to ensure this, you need to enable and disabled fields and buttons at the appropriate times.

A new *Clear* button must also be added to allow the user to reset the form after a record has been fetched.

- 1. Drop a push button onto the form.
  - a. Set the button *Name* and *Caption* to CLEAR.
  - b. Create a *Click* event routine for the CLEAR button.
- 2. Your form should appear like this:

		- 🗆 🛛
Department Code	ABCD	Fetch
Department Description	aAbBcCdDeEfFgGhHiljJ	Insert
Standard Number	1,234,567	Update
	Clear	Delete
aAbBcCdDeEfFgGhHiljJkKlLmN	/nNoOpPqQrRsStTuUvVwWxXyYzZ	•••••

3. To centralize your code, you will create a SUBROUTINE that can be called whenever the form needs to be reset. This subroutine needs to reset all fields to their default values and enable the Fetch and Insert buttons, as well as the DEPTMENT field. The Update and Delete buttons are disabled until a record has been fetched.

Your code should appear as follows:

```
SUBROUTINE NAME(INITFORM)
#FORMDATA := *DEFAULT
#UPDATE.Enabled #DELETE.Enabled := False
#DEPTMENT.Enabled #FETCH.Enabled #INSERT.Enabled := True
ENDROUTINE
```

4. In the *CLEAR.Click* event routine, execute your INITFORM subroutine. Your finished code should appear as follows:

EVTROUTINE HANDLING(#CLEAR.Click) EXECUTE SUBROUTINE(INITFORM) ENDROUTINE

- 5. In the form's Initialize event routine, execute your INITFORM subroutine. EVTROUTINE HANDLING(#com_owner.Initialize) SET COM(#com_owner) CAPTION(*component_desc) EXECUTE SUBROUTINE(INITFORM) ENDROUTINE
- 6. Change the *FETCH.Click* event routine code so that the Department Description field, Fetch and Insert button are disabled, and the Update and Delete buttons are enabled if a record has been successfully retrieved.

Your finished code should appear something like the following:

EVTROUTINE HANDLING(#FETCH.Click) FETCH FIELDS(#DEPTMENT #DEPTDESC) FROM_FILE(DEPTAB) WIT IF_STATUS IS_NOT(*OKAY) MESSAGE MSGTXT('Error retrieving Department') ELSE **#DEPTMENT.Enabled #FETCH.Enabled #INSERT.Enabled := False #UPDATE.Enabled #DELETE.Enabled := True** ENDIF ENDROUTINE

7. Remove the WITH_KEY parameter on the UPDATE and DELETE commands. The commands should appear simply as follows:

UPDATE FIELDS(#FORMDATA) IN_FILE(DEPTAB)

DELETE FROM_FILE(DEPTAB)

Also, after the update or delete has completed successfully, be sure to execute your INITFORMS subroutine.

**FRM035** – Appendix contains a complete sample solution for form **iiiMntDept**.

8. Compile and execute the form.

When the form first appears, only *Fetch*, *Insert* and *Clear* are allowed.

3	Database Maintenance	- • ×
Department Code		Fetch
Department Description		Insert
Standard Number	0	Update
	Clear	Delete

- a. Add some new test data.
- b. *Fetch* one of your newly inserted records.

3	Database Maintenance	
Department Code		Fetch
Department Description	Department III	Insert
Standard Number		Update
	Clear	Delete

The Update and Delete buttons are now enabled while the Fetch and Insert buttons are disabled.

The Department Code is disabled to prevent the user from changing this code. You can now Update the record or Delete it. The Clear button will reset the form so that a different record can be fetched or more records inserted.

9. Close your form and close the form in the editor.

### Summary

#### **Important Observations**

- GROUP_BY commands help you simplify your code.
- You should always check the I/O status after performing database operations. The IF_STATUS command supports *OKAY, *ERROR, *VALERROR, *NORECORD, *ENDFILE, *BEGINFILE, *EQUALKEY, *NOTEQUALKEY.
- Program level validation rules can be added using a BEGINCHECK / ENDCHECK and related validation commands CALLCHECK, CONDCHECK, DATECHECK, FILECHECK, RANGECHECK, VALUECHECK, IF_ERROR, SET_ERROR.
- A record should be fetched from the database prior to updating or deleting the record if you are not using a key to identify the record. For more details about cross-update and delete, please search for *Cross Update* notes in the *Technical Reference Guide*. For example see *UPDATE Comments / Warnings / Understand UPDATE Command*.
- LANSA provides many Built-In Functions that can be used to perform specialized tasks in LANSA. Intrinsic field methods can be used in place of many of these Built in Functions.

# **Tips & Techniques**

- The CHECK_ONLY parameter on file commands allows you to check what would happen if the file operation is performed but the operation is not actually performed. For example, you can use CHECK_ONLY(*YES) on the delete command to check if a record can be deleted from a file without actually performing the delete.
- The ISSUE_MSG parameter on the FETCH commands will only display a message if a record is not found. It is generally better to check the IO\$STS or use the IF_STATUS and then display messages to the user.
- Typically, you do not need to open or close files in LANSA applications. In special circumstances, such as large application with hundreds of files being accessed, you may wish to use the OPEN and CLOSE statements.
- Special values can be used with the database commands such as FETCH, INSERT and UPDATE. For example, the following values can be used with the FETCH command:

- *ALL specifies that all fields from the currently active file be fetched.
- *ALL_REAL specifies that all real fields from the currently active file be fetched.
- *ALL_VIRT specifies that all virtual fields from the currently active file be fetched.
- *EXCLUDING specifies that fields following this special value must be excluded from the field list.
- *INCLUDING specifies that fields following this special value must be included in the field list. This special value is only required after an *EXCLUDING entry has caused the field list to be in exclusion mode.
- It is strongly recommended that the special values *ALL, *ALL_REAL or *ALL_VIRT be used sparingly and only when really required. Fetching fields which are not needed causes the function to retrieve and map fields unnecessarily, invalidates cross-reference details (shows fields which are not used in the function) and increases the Crude Entity Complexity Rating of the function.
- Most of your validation checks should be performed using file level validation rules in the *Repository*. Your program level validations support *ERROR, *NEXT and *ACCEPT processing similar to the repository checks.

### What You Should Know

- How to use the FETCH, INSERT, UPDATE and DELETE database commands.
- How to handle errors using the IF_STATUS command when performing file operations.
- How to code program level validations using BEGINCHECK / ENDCHECK, CONDCHECK, FILECHECK, RANGECHECK, VALUECHECK, IF_ERROR, and SET_ERROR.
- How program validations relate to the repository validation performed by the Object Access Modules.
- How to use a Built-In Function to display a message box with user confirmation.

## FRM035 – Appendix

Your finished code for form **iiiMntDept** should appear something like the following:

FUNCTION OPTIONS(*DIRECT) BEGIN_COM ROLE(*EXTENDS #PRIM_FORM) CLIENTHEIGHT(174) CLIENTWIDTH(500) HEIGHT(212) LEFT(416) TOP(250) WIDTH(516) DEFINE COM CLASS(#DEPTMENT.Visual) NAME(#DEPTMENT) DISPLAYPOSITION(1) HEIGHT(19) LEFT(16) PARENT(#COM OWNER) TABPOSITION(1) TOP(8) USEPICKLIST(False) WIDTH(201) DEFINE COM CLASS(#DEPTDESC.Visual) NAME(#DEPTDESC) DISPLAYPOSITION(2) HEIGHT(19) LEFT(16) PARENT(#COM OWNER) TABPOSITION(2) TOP(32) WIDTH(324) DEFINE_COM CLASS(#PRIM_STBR) NAME(#STBR_1) DISPLAYPOSITION(3) HEIGHT(24) LEFT(0) MESSAGEPOSITION(1) PARENT(#COM OWNER) TABPOSITION(3) TABSTOP(False) TOP(150) WIDTH(500) DEFINE_COM CLASS(#STD_NUM.Visual) NAME(#STD_NUM) CAPTION('Insert') DISPLAYPOSITION(4) LEFT(16) PARENT(#COM_OWNER) TABPOSITION(4) TOP(64) DEFINE COM CLASS(#PRIM PHBN) NAME(#UPDATE) CAPTION('Update') DISPLAYPOSITION(5) LEFT(389) PARENT(#COM OWNER) TABPOSITION(5) TOP(78) DEFINE COM CLASS(#PRIM PHBN) NAME(#DELETE) CAPTION('Delete') DISPLAYPOSITION(6) LEFT(389) PARENT(#COM_OWNER) TABPOSITION(6) TOP(112) DEFINE_COM CLASS(#PRIM_PHBN) NAME(#CLEAR) CAPTION('Clear') DISPLAYPOSITION(7) LEFT(168) PARENT(#COM_OWNER) TABPOSITION(7) TOP(96) DEFINE_COM CLASS(#PRIM_PHBN) NAME(#insert) CAPTION('Insert') DISPLAYPOSITION(8) HEIGHT(26) LEFT(389) PARENT(#COM_OWNER) TABPOSITION(8) TOP(40) DEFINE_COM CLASS(#PRIM_PHBN) NAME(#Fetch) CAPTION('Fetch') DISPLAYPOSITION(9) HEIGHT(26) LEFT(389) PARENT(#COM_OWNER) TABPOSITION(9) TOP(8) GROUP_BY NAME(#FORMDATA) FIELDS(#DEPTMENT #DEPTDESC #STD NUM) DEFINE FIELD(#ANSWER) TYPE(*CHAR) LENGTH(6)

EVTROUTINE HANDLING(#COM OWNER.Initialize) SET COM(#COM_OWNER) CAPTION(*COMPONENT_DESC) EXECUTE SUBROUTINE(INITFORM) **ENDROUTINE** EVTROUTINE HANDLING(#FETCH.Click) FETCH FIELDS(#DEPTMENT #DEPTDESC) FROM FILE(DEPTAB) WITH KEY(#DEPTMENT) IF STATUS IS NOT(*OKAY) MESSAGE MSGTXT('Error retrieving Department') ELSE #DEPTMENT.Enabled #FETCH.Enabled #INSERT.Enabled := False **#UPDATE.Enabled #DELETE.Enabled :=** True **ENDIF ENDROUTINE** EVTROUTINE HANDLING(#INSERT.Click) BEGINCHECK FILECHECK FIELD(#DEPTMENT) USING FILE(DEPTAB) USING_KEY(#DEPTMENT) FOUND(*ERROR) NOT_FOUND(*NEXT) MSGTXT('Department Code already exists.') VALUECHECK FIELD(#DEPTDESC) WITH_LIST(NONE END LAST) IN_LIST(*ERROR) NOT_INLIST(*NEXT) MSGTXT('This description is reserved.') RANGECHECK FIELD(#STD_NUM) RANGE((1 10)) MSGTXT('Must be in range 1 to 10.') CONDCHECK FIELD(#DEPTMENT) COND(#DEPTMENT.Contains( ' ' )) IF TRUE(*ERROR) IF FALSE(*NEXT) MSGTXT('Code cannot contain embedded blanks.') IF COND(#DEPTMENT *EQ #DEPTDESC) SET ERROR FOR FIELD(#DEPTMENT #DEPTDESC) MSGTXT('Department Code cannot be the same as Department Description.') **ENDIF ENDCHECK** INSERT FIELDS(#FORMDATA) TO FILE(DEPTAB) IF STATUS IS(*OKAY) MESSAGE MSGTXT('Department inserted successfully') **#FORMDATA := *DEFAULT** ELSE IF STATUS IS(*ERROR) MESSAGE MSGTXT('Error inserting Department')

**ENDIF ENDIF ENDROUTINE** EVTROUTINE HANDLING(#UPDATE.Click) UPDATE FIELDS(#FORMDATA) IN FILE(DEPTAB) IF_STATUS IS(*OKAY) MESSAGE MSGTXT('Department updated successfully') EXECUTE SUBROUTINE(INITFORM) ELSE IF_STATUS IS(*NORECORD) MESSAGE MSGTXT('Department not found') **ELSE** IF_STATUS IS(*ERROR) MESSAGE MSGTXT('Error updating Department') **ENDIF ENDIF ENDIF ENDROUTINE** EVTROUTINE HANDLING(#DELETE.Click) USE BUILTIN(MESSAGE_BOX_SHOW) WITH_ARGS(YESNOCANCEL NO QUESTION *COMPONENT 'Are your sure you want to delete?') TO_GET(#ANSWER) IF COND(#ANSWER = YES) DELETE FROM_FILE(DEPTAB) IF_STATUS IS(*OKAY) MESSAGE MSGTXT('Department deleted successfully') EXECUTE SUBROUTINE(INITFORM) ELSE IF STATUS IS(*NORECORD) MESSAGE MSGTXT('Department not found') ELSE IF STATUS IS(*ERROR) MESSAGE MSGTXT('Error deleting Department') **ENDIF ENDIF ENDIF ENDIF ENDROUTINE** EVTROUTINE HANDLING(#CLEAR.Click)

EXECUTE SUBROUTINE(INITFORM) ENDROUTINE SUBROUTINE NAME(INITFORM) #FORMDATA := *DEFAULT #UPDATE.Enabled #DELETE.Enabled := False #DEPTMENT.Enabled #FETCH.Enabled #INSERT.Enabled := True ENDROUTINE END_COM

# FRM045 - Using LANSA Debug

### **Objectives:**

- To introduce the Visual LANSA debugger.
- To learn how to set breakpoints in a form.
- To learn how to display and change variables in a program.

The focus of this exercise is how to use the features of the debugger. A detailed understanding of the RDML commands is not important at this time. The Programming exercises teach the basic coding practices.

To achieve these objectives you must complete the following:

Step 1. Execute Applications with Debug

Step 2. Debug Features

Step 3. Set Breakpoints

Step 4. Display/Change Variables

Step 5. Set Breakpoint Properties

Step 6. Set a Break on Value

Summary

### **Before You Begin**

You may wish to review the following topic in the *Visual LANSA User Guide*: Debugging Applications.

In order to complete this exercise, you must have completed the previous exercises.

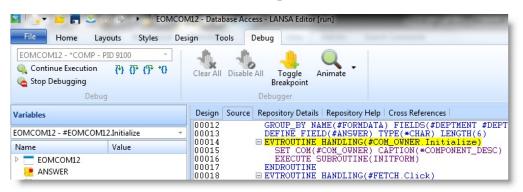
## **Step 1. Execute Applications with Debug**

In this step you will execute your Department Maintenance form (iiiMntDept) which you created in the previous exercise, using the debugger. The form must have been compiled with debug enabled.

- 1. If your form iiiMntDept is open in the editor, close it.
- 2. Using the *Repository* tab, locate your form iiiMntDept.
- 3. Right-click the iiiMntDept form to display the context menu, and select the *Debug / Start* option.

the second second second	iiiMntDept Da iiiSettingsDialog Set	10	Open	1	
J K		-	Compile Execute		
M			Debug >	9	Start
N		×	Delete from Repository		Debug Disable
P			Find		Debug Disable All

4. As soon as the form is executed, it is opened in the editor:



- 5. The first executable line is displayed and the debugger is waiting for your commands. As you can see from the above image, the form is about to execute the *Initialize* event handling routine. Notice this line is highlighted in yellow. This is the line which is about to be executed.
- 6. Notice that the *Debug* tab of the ribbon is displayed with a set of debug commands. You can position the cursor over these commands to display the tool tips.

File Home Layo	outs Styles	Design	Tools	Debug		
EOMCOM12 - *COMP - PI	D 9100 - ि 0 + 0 + *0	Clea	ar All Disak		oggle akpoint	Animate -
Debug				Debug	ger	
Variables	Step Into	De	sign Sourc	e Reposi	tory Detail	s Repository H
EOMCOM12 - #EOMCOM12	Step Into (F8)	- 00	)12 )13	DEF	INE FIE	AME(#FORMD, LD(#ANSWER
Name	Value		014 015			HANDLING(; #COM_OWNER

7. Also, you should notice that a new Variables tab has appeared. This tab lists the variables used in the form.

Variables	M12 Initialize	
Name	Value	
<ul> <li>EOMCOM12</li> <li>ANSWER</li> <li>DEPTDESC</li> <li>DEPTMENT</li> <li>IO\$MDE</li> <li>IO\$MSGTXT</li> <li>IO\$STS</li> </ul>	DIS	
e std_num	0	

### **Step 2. Debug Features**

In this step you will simply review some of the basic debug features.

1. Select the *Breakpoints tab* on the bottom of the editor. If necessary open the *Breakpoints* tab from the *Home* ribbon, *Views* menu.

You have no break points set at this time so no items are listed:

X 🔥 🖕 🚈	
) Assistant   💽 Call Stack  🧃 Breakpoints 🕹 Compile	🕜 Help

2. Use the *Views* menu to open the *Call Stack* tab at the bottom of the editor:

× ₽	*COMPONENT *COMPONENT - *COMP
1	IIIMNTDE - #IIIMNTDE.Initialize
	🛛 Call Stack 🛛 🧊 Breakpoints 🛛 迭 Compile 🛛 🛼 Text Search 🛛 👘
	Ln 20, Col 1

The *Call Stack* lists all programs (processes, functions, components and reusable parts) in the order that they have been invoked.

3. Press the F8 (Step Into) key once, and you will see that the highlighted line in the editor advances to the next line to be executed (SET command). This command has not been executed yet.

## Step 3. Set Breakpoints

In this step you will learn how to set breakpoints in a form. A breakpoint is a location where the form will stop execution so that you can review the code or variables. You will add a breakpoint at the beginning of the UPDATE button click event, before any data is written to the file.

1. Scroll down to the event handling routine to UPDATE.Click

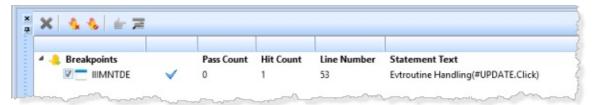
**Note:** You can rapidly move to any routine in your code, by using the *Go To* tab. Expand routines and click on any one to move to that line in the editor.

2. Press the F9 key to set a breakpoint on this line. The line is highlighted in red.

Design	Source Repository Details Repository Help Cross References
00039	MESSAGE MSGTXT('Department inserted successfully')
00040	#FORMDATA := *DEFAULT
00041	ELSE
00042	□ IF_STATUS IS(*ERROR)
00043	MESSAGE MSGTXT('Error inserting Department')
00044	ENDIF
00045	ENDIF
00046	ENDROUTINE
00047	EVTROUTINE HANDLING(#UPDATE.Click)
00048	UPDATE FIELDS(#FORMDATA) IN_FILE(DEPTAB)
00049	□ IF_STATUS IS(*OKAY)
00050	MESSAGE MSGTXT('Department updated successfully')
00051	EXECUTE SUBROUTINE(INITFORM)
00052	ELSE

**Note:** You can set break points both before executing Debug and while running in Debug. Breakpoints are remembered in both cases (they are saved when you close the form in the editor).

- 3. Press F5 to execute the form. Enter ADM as the department code and click the Fetch button. The form will display the Administration department. Leave the value of the Department Description as it is and click the Update button. When the form reaches the breakpoint, source code will be redisplayed.
- 4. Display the Breakpoints tab. The new breakpoint will be listed as follows:



The program will now stop before executing this line of code. This will allow you to view and change the variables or make other changes to debug settings.

- 5. Scroll to the top of the code and set the focus on the GROUP_BY command and press F9. Notice that this command is not highlighted. You can only set breakpoints on executed lines of code. The GROUP_BY is a definition used for compilation and is not used at execution time.
- 6. Press the F5 key to execute the application until the next breakpoint. Since there are currently no more breakpoints, the UPDATE will be processed and the form redisplayed.

# **Step 4. Display/Change Variables**

In this step you will learn how to display and change the variables in a program as it is executing.

- 1. Enter a department code of **ADM** and click the Fetch button. When the Administration department is displayed, leave the values unchanged and click the Update button. Once again the form will stop at the start of the UPDATE.Click event and display the source in the editor.
- 2. Display the Variables tab to see a list of the variables in the function. It will appear something like the following:

Name	Value	
<ul> <li>IIIMNTDE</li> <li>ANSWER</li> <li>DEPTDESC</li> <li>DEPTMENT</li> </ul>	Administration ADM	
IO\$MDE	DIS	
IO\$MSGTXT IO\$STS	ок	
📒 STD_NUM	0	

3. When debugging components (forms, reusable parts and WAMs) you need to be aware that many fields are components. Note that DEPTDESC and DEPTMENT are shown as type Component. If you open the context menu (right mouse click) on one of these, you will not be able to change the value:

IIMNTDE - #UPDATE.CI	ick	-
Name	Value	
ANSWER DEPTDESC	Administration	
DEPTMENT	ADM	
IOSMDE	Set Value	
IO\$MSGTXT IO\$STS STD_NUM	Break On Value Condit View Breakpoint	tion
	Hexadecimal Display Ancestor Features	

Note that on the image above, the Set Value option is not available (grayed out).

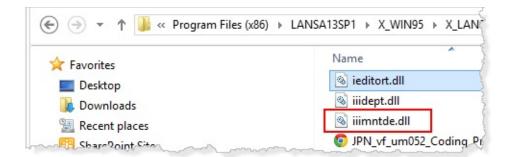
Try right clicking on the ANSWER variable (which is a work field value in the form) and note that you could change this value.

4. To change the value of DEPTDESC you need to expand the form component **iiiMntDept** and then expand the component DEPTDESC and select **Value** as shown:



Note that the Set Value option is available.

Notice that the *Variables* tab uses the form's *Identifier* to name the component, in this case **IIIMNTDE**. The form compiled object is a Windows DLL named **iiimntde.dll**, as shown:



In a later exercise, this topic will be covered in more detail. In most situations your own code can refer to the form using its *Long Name* and LANSA accesses the object using it's *Identifier*.

5. Select Set Value and the Set Value dialog appears, enter a value of NEW ADMINISTRATION.

Set Value	9
New Administration	ОК
	Cancel

Click OK and note that the new value is shown in the Variables tab.

6. Press *F5* to run the application. The form will be redisplayed. Enter ADM in the department code and click the Fetch button.

Department Code	ADM	Fetch
Department Description	New Administration	Insert
Standard Number	0 ¢	Update
	Clear	Delete

The department record has been updated with the changed value.

#### **Step 5. Set Breakpoint Properties**

1. Close your Department Maintenance form and switch to the VL editor. Add a BEGIN_LOOP/END_LOOP at the beginning of the UPDATE.Click event handling routine.

Loop 10 times using STD_COUNT as the loop count and display a message 'Message number is nn' containing STD_COUNT. Your code should look like the following:

Evtroutine Handling(#UPDATE.Click)

Begin_Loop Using(#STD_COUNT) To(10)

Message Msgtxt('Message number is ' + #std_count.asstring)

End_Loop

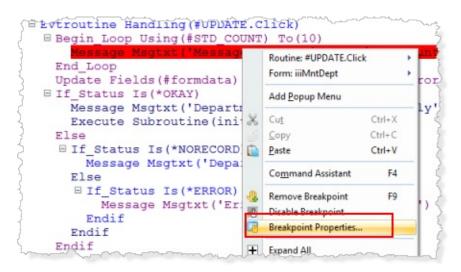
Update Fields(#FORMDATA) In_File(DEPTAB) Val_Error(*NEXT)

. . . . .

- This provides a line of code which executes a number of times, each time the UPDATE.Click event routine runs.
- 2. Compile your form.
- 3. Run the form in debug. When the program stops at the Initialize routine, scroll down to the UPDATE.Click event handling routine and clear the breakpoint on the UPDATE statement.

There are a number of ways you can do this. For example:

- a. select this line of code and press *F*9 or use the Toggle Breakpoint button on the *Debug* ribbon.
- b. select this line of code and use the right mouse menu option, *Remove Breakpoint*.
- c. display the *Breakpoints* tab, select this breakpoint and use the *Remove Breakpoint* toolbar button.
- 4. Select the MESSAGE command inside the loop and press *F9* to set this as a breakpoint.
- 5. Use the right mouse menu option while selecting the MESSAGE command to show the *Breakpoint Properties* dialog.



	Breakpoint Propertie	
ieneral		ОК
Pass count:		Cancel
3		÷
Break for o	current instance only	

- 6. Set a Pass count of 3. During an update, the loop will now execute twice and break on the third execution of the MESSAGE command.
- 7. Press *F5* to execute the form, Fetch a department and press the Update button. Debug should break when the MESSAGE command executes the 3rd, 6th and 9th time.
- 8. Close your form.
- 9. Remove the breakpoint from the MESSAGE command.

### Step 6. Set a Break on Value

In this step you are going to again use the UPDATE.Click event routine and set a breakpoint on the first IF_STATUS command, with a break on value setting for variable IO\$STS. This is a 2 character I/O status, returned by the UPDATE command. When a validation error occurs it will contain 'VE'.

1. Switch to the editor and review the definition of the UPDATE command in the UPDATE.Click event routine. If it looks like the following:

Update Fields(#FORMDATA) In_File(DEPTAB)

Then because the VAL_ERROR parameter for the FETCH command, has a default value of *LASTDIS, which in a form means branch to the end of the routine, the IF_STATUS which follows it is never executed when a validation error occurs. Ensure your update command looks like the following:

Update Fields(#FORMDATA) In_File(DEPTAB) Val_Error(*NEXT) Recompile your form if you needed to make a change.

- 2. Set a breakpoint on the first IF_STATUS command.
- 3. Run the form in debug. When the editor is displayed at the Initialize routine, scroll down to the UPDATE.Click event routine and set a break point on the first IF_STATUS command.
- 4. In the variables tab, select the IO\$STS variable and use the right mouse menu to *Break on Value Condition*

Variables		Design
IIIMNTDE - #IIIMNTDE.Init	ialize 🔹	2
Name	Value	
<ul> <li>IIIMNTDE</li> <li>ANSWER</li> <li>DEPTDESC</li> <li>DEPTMENT</li> <li>IOSMDE</li> <li>IOSMSGTXT</li> <li>IOSSTS</li> </ul>	DIS	
STD_COUNT	Set Value	
STD_NUM	Break On Value Condition	
	View Breakpoint	1
	Hexadecimal Display Ancestor Features	

5. In the *Breakpoint Properties* dialog, select the *Value* tab and set the breakpoint to break when IO\$STS is equal to 'VE'

Breakpoint Propert	ies
General Value	ОК
<ul> <li>Break</li> <li>Is equal to</li> </ul>	Cancel
Is not equal to	
Is less than	
Is greater than	
VE	

- 6. Select *OK* and press *F5* to run the form.
- 7. Fetch a department and press Update. Debug should not break because IO\$STS is not equal 'VE'
- 8. Fetch a department and clear the department description field and press Update. Debug should now break on the IF_STATUS command.
- 9. Close your form and close it in the editor. You have completed this exercise.

### Summary

#### **Important Observations**

- Using the Visual LANSA editor, you can interactively debug functions, forms, reusable parts, WAMs and web event functions locally.
- You can also remotely debug RDMLX functions when they are running on the server, including web events and WAMs using the VL Editor
- Programs must be compiled 'debug enabled' in order to use debug.

# **Tips & Techniques**

- By default, debug stops at the first executable line of code, controlled by *Debug* settings in the *Editor Options* dialog.
- Breakpoints are saved with the source code
- Breakpoint features include variables, breakpoint and call stack tabs
- Variables and component values can be viewed and edited
- Breakpoint properties support break on condition and pass count.

# What You Should Know

• How to use debug for local debugging.

## FRM055 - List Component Basics

### **Objectives:**

- To learn how to use a combo box and list view to show data
- To learn how to use the SELECT command to read multiple records from a file
- To learn how to add entries using the ADD_ENTRY command
- To learn about list properties and how lists can be made to interact with on another
- To learn how to define and execute a subroutine using the SUBROUTINE/ENDROUTINE and EXECUTE commands
- To create form which shows displays department and section tables in combo boxes and employee data in a list view.
- To show how to use Visual Styles and Themes to change the appearance of forms and their components.

Department Administrator Dept		Section			
				-	
Employee N	Employee Su	Employee Gi	Post / Zip Co	Employee Sal	
A1001	Jones	Shirley	2001	2,345.82	
A1012	Paul	Patrick	2147	26,456.04	
A1013	Pattinson	George	2016	78,977.04	
A1015	Woods	Bradley	2030	313,000.04	
A1020	Douglas	Adam	2147	121,500.04	
A1021	McCully	Lisa	2153	87,000.04	
A1025	Robinson	Mary	2005	44,455.04	
A1027	Morrison	Alan	2007	1,878,773.04	
A1111	Verey	Warren	2345	45,678.04	
					-

- The completed form loads departments and related sections into combo boxes and populates a list view with employees for the current section. Initially the first department and first section in a department are selected.
- When a new department is selected, the section combo box and list view are

repopulated

• When a new section is selected, the employee list view is cleared and repopulated.

To achieve these objectives you must complete the following:

Step 1. Create a Simple List

Step 2. Select Data to Fill the List

- Step 3. Create Multiple Lists
- Step 4. Fill the Lists
- Step 5. Make List View Columns Sortable
- Step 6. Change Appearance of the Form
- Step 7. Read Sorted List Items (optional)

Step 8. Sort Department and Sections Combo Boxes (optional)

Summary

# **Before You Begin**

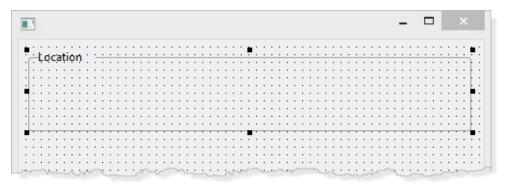
• You must have completed the previous exercises.

# Step 1. Create a Simple List

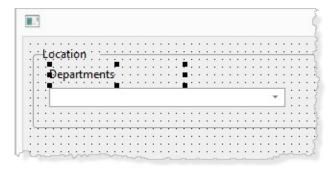
In this step you will begin by creating a simple form containing a combo box containing a list of all departments. A combo box (or drop down), is a simple list component which displays one column with one entry visible. The list may be expanded by clicking the drop down button .

When you select a department its details (DEPTMENT and DEPTDESC) will be displayed on the form.

- 1. Create a *New Form / Basic Form iiiListBasics List Basics* (where iii are your course assigned initials).
- 2. Drag a *Group Box* onto the top of the form from the *Controls* tab, resize it and set its *Caption* to **Location**. Your form should look like the following:



- 3. Drop a *Combo Box* onto the *Group Box*. Extend its length to approximately half the form width. Change its *Name* to **DEPTS**. Move the *Combo Box* towards the bottom edge of the Group Box. Remember as well as dragging with the mouse, you can move components by holding down the Control key and using the cursor keys.
- 4. Drop a *Label* component above the *Combo Box*, resize it and change its *Caption* to **Department**. Your form should look like the following:



5. Display the *Source* tab. You will see DEFINE_COM like the following:

Define_Com Class(#PRIM_GPBX) Name(#GPBX_1) Caption('Location') Dis Define_Com Class(#PRIM_CMBX) Name(#DEPTS) Componentversion(1) D

Note the combo box DEPTS has a parent of GPBX_1. The group box is a "container". The combo box DEPTS belongs to this container.

- 6. Add columns to the DEPTS Combo Box. A combo box displays one column, but may contain other hidden columns.
  - a. Drop field DEPTMENT into the combo box and use the *Details* tab to set its *Visible* property to **False**. To do this, select the component CBCL_1 from the drop down at the top of the *Details* tab. You are then working with the first column component for the combo box DEPTS.

BCL_1			
🗉 🕶 🦞			
Properties Events Methods			
ComponentClassName	PRIM_CBCL		
ComponentPatternName	PRIM_CBCL		
ComponentTag			
ComponentTypeName	PRIM_CBCL		
DisplayPosition	0		
🖻 Name	CBCL_1		
Cwner 🖉	#EOMCOM13		
Parent 🖉	#Depts		
SortDirection	Ascending		
SortPosition	0		
🖉 SortType	Word		
P Source	#DEPTMENT		
Picklist 🖉	False		
🖻 Visible	False	-	
🖉 Width	20	3	
P WidthType	Scaleable		

b. Drop the field DEPTDESC into the DEPTS combo box.Your form should now look like the following:



7. Display the Source tab. You will see new DEFINE_COM statements for the columns, like the following:

Define_Com Class(#PRIM_CBCL) Name(#CBCL_1) Parent(#DEPTS) Source Define_Com Class(#PRIM_CBCL) Name(#CBCL_2) Displayposition(1) Pare

The *Combo Box* columns are automatically named. They have a parent of DEPTS. The columns belong to the combo box DEPTS.

They have a *Source property*, DEPTMENT, for example, meaning the column is based on field DEPTMENT.

8. Drop fields DEPTMENT and DEPTDESC onto the form below the *Group Box*. Lengthen the form first if necessary. Your form should look like the following:

B		-		×
Location		 		
Department		 · · ·		
aAbBcCdDeEfFgGhHiljJ	::::	 :::	: : : :	::::
Department Code ABCD		 		
Department Description : : aAbBcCdDeEfFgGhHiljJ		 		
· · · · · · · · · · · · · · · · · · ·		 		• • • • •

## Step 2. Select Data to Fill the List

In this step you will add code in the form *Initialize* event to populate the DEPTS combo box.

1. Your code to add entries from the file DEPTAB to the DEPTS combo box should use the CLR_LIST, SELECT, ADD_ENTRY and ENDSELECT commands.

SELECT / ENDSELECT is a database I/O command which reads records based entry sequence, by full key, by partial key or generically for example. Refer to the *Technical Guide* for further details and examples. Remember you can press *F1* on any command in the editor to jump straight to its online guide entry.

Your code should look like the following:

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

**Note:** the combo box definition DEPTS can be used in the SELECT Fields() parameter, to retrieve the fields defined as columns in DEPTS.

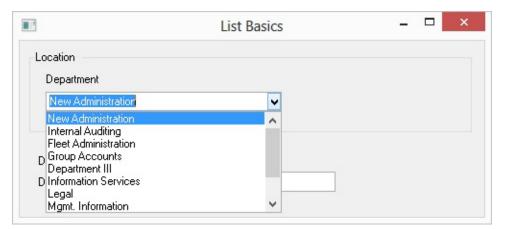
2. Compile and test your form.

3	List Basics	- 🗆 ×
Location		
Department		
New Administration	¥	
Department Code	ADM	
Department Description	New Administration	

**Note:** The first entry in the combo box has focus and the fields on the form are populated. When a row in a list component has focus or is selected, the

form fields for the list columns are populated.

3. Click on the drop down button to see all list entries:



The list is displayed in the sequence that entries were loaded. In this case it is loaded in department code sequence (DEPTMENT). We will look later at how entries in a list can be sorted. In this example one solution could be to add a logical file to the file DEPTAB in DEPTDESC sequence and use this logical file to load the combo box.

**Note:** the dropped down view has a scroll bar. The *DropDownCount* property of the combo box can be changed to display more entries if required. The default value is **8**.

4. Select a different department. Once again note that the form fields are populated from the selected list row.

# **Step 3. Create Multiple Lists**

In this step you will add a second combo box which will contain the Sections for the current department and a List View component, which will contain employee data for the selected section.

- 1. Drop a second *Combo Box* onto the *Group Box*, alongside the department combo box. Position and resize it. Change its *Name* to SECTS.
- 2. Drop a *Label* component above the new *Combo Box*. Position and resize it. Change its *Caption* to Sections. Remember you can use the *Align* dialog on the *Design* tab to align components. Your form should now look like the following:

				:::					:
Location									
Department		Sectio	ns	:::		:::		::	:
· · · · · · · · · · · · · · · · · · ·									
<ul> <li>aAbBcCdDeEfFgGhHiljJ</li> </ul>	*	• · · [						+	
• • • • • • • • • • • • • • • • • • • •									1
						• • •		• •	
									:
Department Code	· · · ·					• • •			•
Department Code ABC						:::		: :	:
Densitiment Description	BcCdDeEfFgG	h Hili I							
. Department Description aAu	becaberingo	in ngo			• • •	• • •	• • •	• •	•
									:

3. On the *Repository* tab, find the file SECTAB and expand it. Drag and drop field SECTION into combo box SECTS. Select the *Details* tab. Note that you are working with **CBCL_3**, the first column added to SECTS, based on field SECTION. Change its *Visible* property to False.

Drag and drop field SECDESC into combo box SECTS. Section description will be the visible column in SECTS.

- 4. Delete the fields DEPTMENT and DEPTDESC from the form.
- 5. Lengthen the form and drag and drop a List View onto the lower form area. Change its *Name* to **EMPLOYS**. Resize it to occupy all the space. Your form should look like the following:



6. Locate the file PSLMST on the *Repository* tab and expand it. Drag and drop fields, EMPNO, SURNAME, GIVENAME, POSTCODE and SALARY into the list view EMPLOYS. Your form should look like the following:

Department			Sections	
aAbBcCdDeE	tFgGhHiljJ		aAbBcCdDeEfFg	GhHiljJ -
· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·		· · · · · · · · · · · · · · · ·	
mployee Nu	Employee Sur	Employee Giv	Post / Zip Code	Employee Sal
BCDE	aAbBcCdDeE	aAbBcCdDeE	123456	123,456,789.12

# Step 4. Fill the Lists

#### **Important Concept**

- Sections belong to a department. If a new department is selected, the SECTS combo box needs to be rebuilt.
- Employees belong to a section. If a new section is selected, the EMPLOYS list view needs to be rebuilt.
- 1. Create a subroutine (SUBROUTINE / ENDROUTINE) to build, rebuild the SECTS combo box. Give the subroutine a Name of **SECTS**. Once again your logic should use the CLR_LIST, SELECT, ADD_ENTRY, ENDSELECT commands. This time your SELECT needs to read the file SECTAB with a key of DEPTMENT.

A subroutine can be defined anywhere in your form code, but not inside an event or method routine.

#### Note:

- File SECTAB is keyed on DEPTMENT and SECTION. You need to select records using the high level key DEPTMENT only.
- You should now be aware that a selected entry in combo box DEPTS will automatically populate the variable DEPTMENT in the form code.

Your code should look like the following:

Subroutine Name(SECTS) Clr_List Named(#SECTS) Select Fields(#SECTS) From_File(sectab) With_Key(#deptment) Add_Entry To_List(#SECTS) Endselect Endroutine

2. A subroutine is executed using the EXECUTE command. Add code at the end of the form Initialize event routine to execute SECTS. Your code should now look like the following:

Evtroutine Handling(#com_owner.Initialize) Set Com(#com_owner) Caption(*component_desc) Clr_List Named(#DEPTS) Select Fields(#DEPTS) From_File(deptab) Add_Entry To_List(#DEPTS) Endselect Execute Subroutine(SECTS) Endroutine

- 3. The SECTS combo box will need to be rebuilt if a new department is selected in DEPTS. Create an ItemGotSelection event routine for DEPTS. Select the DEPTS combo box and do this from the *Details / Events* tab or use the context menu on the DEPTS combo box.
- Execute subroutine SECTS in this event routine. Your code should look like the following:

Evtroutine Handling(#DEPTS.ItemGotSelection) Options(*NOCLEARMESS/ Execute Subroutine(SECTS) Endroutine

4. Compile and test your form. Notice that when the form initially loads, the first section is not displayed although the SECTS combo box contains the correct entries.

		List Basics		_ 🗆 ×
Location Department			Sections	
New Administra	ation	~		V
Imployee Num	Employee Surn	Employee Give	Post / Zip Code	Employee Salary

Notice also that if you select a different department, the SECTS combo box contains the correct entries but continues to display the last selected entry, until a valid entry is selected in SECTS.

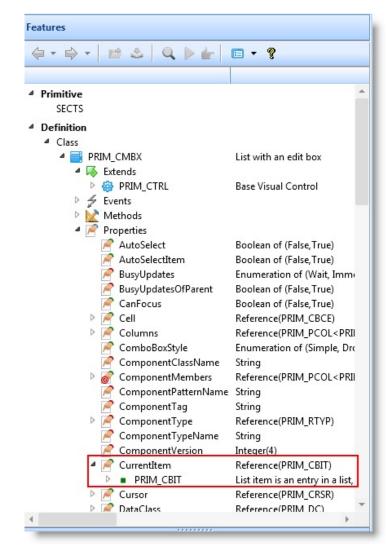
5. Correct this behavior by adding the following code to the end of the SECTS subroutine:

```
Get_Entry Number(1) From_List(#SECTS)
#SECTS.currentItem.focus := true
```

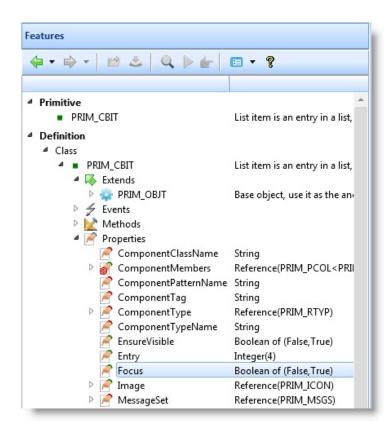
The GET_ENTRY will retrieve a specific entry number from a list. Usually it is executed after logic which locates an entry, for example LOC_ENTRY. See a later exercise for an example of this. In this particular case you are positioning to the top of the SECTS list.

**CurrentItem** is the current row in the list component. **CurrentItem.focus** is the focus property of the current list item.

To discover these properties for any component use *F2 Feature* help for that component, in this case combo box SECTS

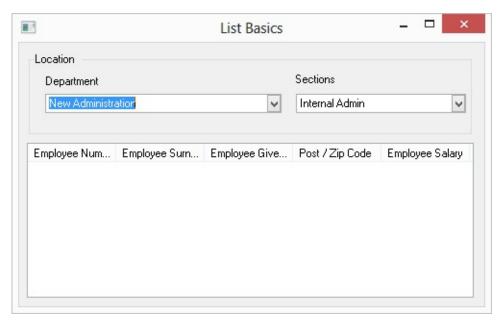


Expand the CurrentItem by double clicking on the list item, PRIM_CBIT.



Expand the **list item** properties to find the Focus property.

6. Compile and retest your form. When it initially loads it should now look like the following:



Try selecting a new department. The first section in the list will always be displayed.

7. When it initially loads, the form should load all three lists

3		List Basics			×
Location					
Department			Sections		
New Administr	ation	~	Internal Admin		¥
Employee Num	Employee Surn	Employee Give	Post / Zip Code	Employee Salary	^
A1001	Jones	Shirley	2001	2,345.82	
A1012	Paul	Patrick	2147	26,456.04	
A1013	Pattinson	George	2016	78,977.04	
A1015	Woods	Bradley	2030	313,000.04	
	Develop	A 1	2147	121,500.04	
A1020	Douglas	Adam	2147	121,000.04	
	Douglas McCully	Adam Lisa	2147 2153	87,000.04	
A1020 A1021	-				
A1020 A1021 A1025	McCully	Lisa	2153	87,000.04	
A1020	McCully Robinson	Lisa Mary	2153 2005	87,000.04 44,455.04	

Create subroutine EMPLOYS. Add logic to this subroutine to clear the list EMPLOYS, select from logical file PSLMST1 with a key of DEPTMENT and SECTION and add entries to the EMPLOYS list view. Your code should look like the following:

```
Subroutine Name(EMPLOYS)
Clr_List Named(#EMPLOYS)
Select Fields(#EMPLOYS) From_File(pslmst1) With_Key(#deptment #section
Add_Entry To_List(#EMPLOYS)
Endselect
Endroutine
```

8. Add an execute subroutine EMPLOYS to the end of subroutine SECTS. The EMPLOYS list will be rebuilt every time the SECTS combo box is rebuilt. Your code should look like the following:

```
Get_Entry Number(1) From_List(#SECTS)
#SECTS.currentItem.focus := true
```

Execute Subroutine(emPLOYS)

Endroutine

 9. Create an ItemGotSelection event routine for combo box SECTS. Add code to execute subroutine EMPLOYS. Your code should look like the following: Evtroutine Handling(#SECTS.ItemGotSelection) Options(*NOCLEARMESS/ Execute Subroutine(EMPLOYS) Endroutine

10. Compile and test your form.

- When the form initially loads, all three lists should be populated.
- When a new department is selected, the SECTS list and EMPLOYS list should be rebuilt
- When a new section is selected, the EMPLOYS list should be rebuilt. Note that there may be some sections for which no employees exist.

# Step 5. Make List View Columns Sortable

In this step you will change the *SortOnClick* property for the surname, postcode and employee number columns in the EMPLOYS list view to allow sorting using these columns.

1. In the *Design* view, select the list view EMPLOYS and click on the Surname column heading. Select the *Details* tab, which will show the properties for this columns (LVCL_2). Change the *SortOnClick* property to **true**.

Details			
LVCL_2		-	
E • ?			
Properties Events Methods			
ReadOnly	True	-	
🖉 SortAsColumn	*NULL		
SortDirection	Ascending		
SortOnClick	True		
SortPosition	0	43	
P SortType	Word		
P Source	#SURNAME		
UsePicklist	False		
🖉 Visible	True		
🖉 VisualStyle	*NULL	Ξ	
🖉 Width	20		
WidthType	Scaleable		
		-	

- 2. Click on the column heading for Postcode and change its *SortOnClick* property to **true**.
- 3. Click on the Employee Number column heading and change its *SortOnClick* property to **true.**
- 4. Compile your and retest your form. You should be able to sort the list of employees by surname, post code or employee number by clicking on the column. Note that by default the column heading shows sort direction

Location					
Department			Sections		
New Administ	ration	~	Internal Admin		~
Employee Num	Employee Surn	Employee Give.	Post / Zip C 🔺	Employee Salary	^
A1001	Jones	Shirley	2001	2,845.82	1
A1025	Robinson	Mary	2005	44,455.04	
A1027	Morrison	Alan	2007	1,878,773.04	
51021	Pattinson	George	2016	78,977.04	
	T duinson				
A1013	Woods	Bradley	2030	313,000.04	
A1013 A1015			2030 2060	31 <mark>3,000.04</mark> 10,00	
A1013 A1015 A1509	Woods	Bradley			
A1013 A1015 A1509 A1404	Woods Redford	Bradley Robert	2060	10.00	
A1013 A1015 A1509 A1404 A1012 A1020	Woods Redford Black	Bradley Robert Gillian	2060 2090	10.00 12,345.04	ļ

# **Step 6. Change Appearance of the Form**

In this step you will change the appearance of the form.

1. Add the following code to the *Initialize* event on your form. It will change the form's appearance using the Office2007Blue theme.

#sys_appln.theme := 2007Blue
#sys_appln.ThemedForms := true

2. Compile and execute your form. Notice the blue theme.

3		List Basics			
Location					
Department			Sections		
New Administr	ation	-	Internal Admin		Ŧ
Englauge Num	Employee Cum	Employee Cive	Bash / Zie Cada	Employee Colony	~
Employee Num	Employee Surn	Employee Give	Post / Zip Code	Employee Salary	<u> </u>
A1001	Jones	Shirley	2001	2,345.82	
A1012	Paul	Patrick	2147	26,456.04	
A1013	Pattinson	George	2016	78,977.04	
A1015	Woods	Bradley	2030	313,000.04	
	Douglas	Adam	2147	121,500.04	
A1020	McCully	Lisa	2153	87,000.04	
A1020 A1021			2005	44,455.04	
A1021		Marv	ZUUD		
A1021 A1025	Robinson	Mary Alan			
A1021		Mary Alan Warren	2005 2007 2345	1,878,773.04 45,678.04	

3. Close your form.

# **Step 7. Read Sorted List Items (optional)**

In this step you will add push buttons with Click events which demonstrate how the List View EMPLOYS can be processed to retrieve the sorted list, or the list in its original order. That is, in the order in which it was loaded.

- 1. Lengthen the form and drop a Status bar at the bottom of the form. This will be used to display messages output by the list processing
- 2. Add a push button below the list view.
  - a. Change its *Caption* to **Read Sorted List**.

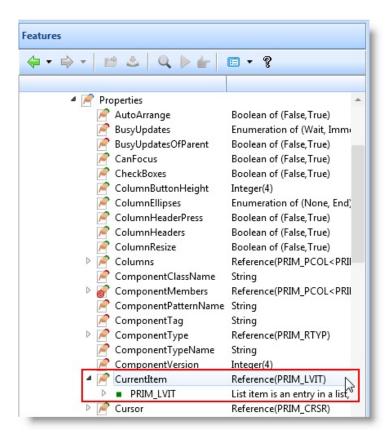
b.Change it's *Name* to **SORTED**.

- c. Create a Click event routine for the SORTED push button.
- d. Add the following code to the SORTED.Click event handling routine:

For Each(#item) In(#EMPLOYS.items) Change Field(#STD_NUM) To(#item.entry) Get_Entry Number(#STD_NUM) From_List(#EMPLOYS) Message Msgtxt('Employee ' + #EMPNO + ' ' + #Surname + ' ' + #givename) Endfor

The FOR loop will read all items in the list EMPLOYS. That is, all rows, in sorted order.

You can use *F2 Feature* help on the List View component to discover all its Properties, Events and Methods:

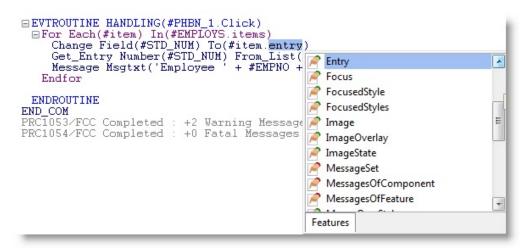


The **Item** object defined by the For Each(#Item) . . . . in the above code, has the same Properties, Events and Methods as **CurrentItem**.

You can use *F2 Feature* help on the List View component. Expand the *CurrentItem* property and double click on the PRIM_LVIT object to show the *Properties, Events* and *Methods* for *CurrentItem*:



Note also that you can use the *Auto Complete* prompter (Ctrl + Space if the prompter is currently turned off in *Editor Settings*) in the editor to discover the properties for Item:



- 3. Add a second push button below the EMPLOYS list view.
  - a. Change its *Caption* to **Read Unsorted List**.
  - b. Change its *Name* to **UNSORTED**.
  - c. Create a Click event for UNSORTED
  - d. Add the following code to the UNSORTED.Click event handling routine:

```
Selectlist Named(#EMPLOYS)
Message Msgtxt('Employee ' + #EMPNO + ' ' + #Surname + ' ' + #givename)
Endselect
```

The SELECTLIST reads the List View EMPLOYS in its unsorted sequence. That is, in the order in which the list was loaded.

Your form should look like the following:

				_ □	×
Department aAbBcCdDeEt	fFgGhHiljJ	•	Sections aAbBcCdDeEfFg	GhHiljJ	
Employee Nu	Employee Sur	Employee Giv	Post / Zip Code	Employee	Sal
, a BCDE	aAbBcCdDeE	aAbBcCdDeE	123456	123,456,78	).12
· · · · · · · · · · · · · · · · · · ·	Re	ad Sorted List	Read l	Jnsorted Lis	t;
aAbBcCdDeEfFgGh	HiljJkKlLmMnNo	OpPqQrRsStTuUv	VwWxXyYzZ		-

- 4. Compile and test your form. Sort the list by clicking on the column heading for Surname, Given Name or Post Code.
  - a. Messages output by the Read Sorted List push button, will be in the order currently displayed. Compare employee numbers with the list view.
  - b. Messages output by the Read Unsorted List push button, will always be in the loaded order. In this case in Employee Number sequence.

# Step 8. Sort Department and Sections Combo Boxes (optional)

This step demonstrates how to use the *SortPosition* property of a combo box column to display departments and sections in description sequence. The column also has a *SortDirection* property with a default value of **Ascending**.

- 1. Combo box columns are automatically named CBCL_1. CBCL_2 and so on. Select the column for Department Description (field DEPTDESC) in the DEPTS combo box using either of the following methods
  - a. In the *Design* view, select the *Details* tab.
  - Select a combo box column (CBCL_1, CBCL_2 etc) from the drop down at the top of the *Details* tab.
  - Ensure that you have selected the column for field DEPTDESC (Source property)

3CL_2	
BCL_1	
CBCL_2	N
BCL_3	2
CBCL_4	
Depts Employs	
OMCOM13	
SPBX_1	
PoisplayPosition	1
Name	CBCL_2
Owner	#EOMCOM13
Parent	#Depts
SortDirection	Ascending
SortPosition	0
SortType	Word
Source 2	#DEPTDESC
Contract Con	False
Visible	True
Width 🖉	20

- b. Alternatively, in the *Design* view select the *Outline* tab.
- Expand the combo box DEPTS
- Hover the cursor over each column (CBCL_1 and CBCL_2 in this case) and use the *Tooltip* to discover which column is required

• Click on the column to select it.

Outline
A C EOMCOM13
Employs
4 🗂 GPBX_1
🔺 🧮 Depts
↓ CBCL_1
↓ CB/\1_2
A LABL1
LABL_CBCL_2 - DEPTDESC, Department Description, Alphanumeric(20)
ECTS
PHBN_1
T STBR 1

2. With the column in the combo box DEPTS selected which contains department description (field DEPTDESC). Use the *Details* tab to change its *SortPosition* property to **1**.

Details	
CBCL_2	*
<b>□</b> • ?	
Properties Events Methods	
ComponentClassName	PRIM_CBCL
🖻 ComponentPatternName	PRIM_CBCL
🖻 ComponentTag	
ComponentTypeName	PRIM_CBCL
DisplayPosition	1
🖻 Name	CBCL_2
🖻 Owner	#EOMCOM13
🖻 Parent	#Depts
SortDirection	Ascending
SortPosition	μ Ţ
🖻 SortType	Word
🖉 Source	#DEPTDESC
🖉 UsePicklist	False
🔎 Visible	True
🖻 Width	20
🖉 WidthType	Scaleable

- 3. Find the column for combo box SECTS containing section description (SECDESC) and change its *SortPosition* property to **1**.
- 4. Save your changes.

- 5. Now that the combo box entries are sorted by their description field, you will need to change the method used to position the top entry.
  - a. Add the code shown in highlited (red, italic) to the form *Initialize* routine. This gives the first entry in combo box DEPTS focus and retrieves the first entry. Remember the FOR loop processes the sorted list.

```
Evtroutine Handling(#com_owner.Initialize)
Set Com(#com_owner) Caption(*component_desc)
Clr_List Named(#DEPTS)
Select Fields(#DEPTS) From_File(deptab)
Add_Entry To_List(#DEPTS)
Endselect
For Each(#item) In(#DEPTS.items)
#item.focus := true
#std_num := #item.entry
Get_Entry Number(#std_num) From_List(#DEPTS)
Leave
Endfor
Execute Subroutine(SECTS)
Endroutine
```

- b. In the subroutine SECTS, comment out the two lines shown as commented (beginning with *) below(**Hint:** Use *Ctrl* + *W*).
- c. Add the FOR logic also shown in bold. This will give focus to the first displayed entry and retrieve this entry in order to have the correct section code (field SECTION) to build the list of employees.

```
Subroutine Name(SECTS)
Clr_List Named(#SECTS)
Select Fields(#SECTS) From_File(sectab) With_Key(#deptment)
Add_Entry To_List(#SECTS)
Endselect
* Get_Entry Number(1) From_List(#SECTS)
* #SECTS.currentItem.focus := true
For Each(#item) In(#SECTS.items)
#item.focus := true
#std_num := #item.entry
Get_Entry Number(#std_num) From_List(#SECTS)
Leave
```

#### Endfor

Execute Subroutine(EMPLOYS) Endroutine

- 6. Compile and test your new version of form **iiiListBasics**.
- **Note:** Remember you may have at least one department which you added, which has no sections and no employees.

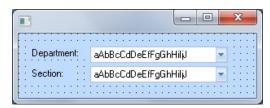
## Summary

### **Important Observations**

- List components such as combo box and list view are defined visually by creating the list and adding the fields (columns) to the list using drag and drop. Lists have many properties which are used to control the sorting and display of the list.
- The SELECT command is used to create a loop structure to read and process multiple records from a file based on a specified selection criteria. The SELECT command can be used with physical and logical files.
- You will usually need to clear the list (CLR_LIST) before populating it. ADD_ENTRY adds an entry to a list component. It creates a new row, with the current values for all fields which make up the list's columns.
- GET_ENTRY positions to a specific row number in a list component. Form variables will be populated.
- #SECTS.CurrentItem.focus := true sets the focus on the current item in the list SECTS. Discover the details of such properties using the *F2 Feature* help.
- There are many events and properties for list components. This exercise provides a simple introduction to using list components
- The sorting of list can be controlled at runtime
- The SUBROUTINE / ENDROUTINE defines a routine which you can EXECUTE from any other routine in your form. Subroutines cannot be defined within an event, method or property routine.
- You can change the appearance of your application using Visual Styles and Themes.
- The FOR loop may be used to read a list in its sorted sequence.
- The SELECTLIST will always read a list in its original loaded order.

# **Tips & Techniques**

• The logic for combo boxes DEPTS and SECTS in this form, could be extended to build a reusable part, which could be used instead of input fields for department and section code, in a form such as iiiCOM11 – Add Employee:



- Using reusable parts is covered in the Windows Applications Workshop
- The ability to put invisible (or hidden) columns into any list is an important concept. In this exercise you want the user to see department description in the combo box, but in the program you need to use the department code (DEPTMENT) as the key field. The need to show the user a description but programmatically interpret their selection from the list as a key is a very common technique.
- List performance is directly related to the number of entries. Ensure you give users sufficient filtering capabilities, so they do not have to create large lists.
- When testing lists, ensure you include using volumes of entries which reflect your production database.

# What You Should Know

- How to define a list component and its columns
- How to fill a list
- How to use events such as ItemGotSelection
- How data is mapped into and out of list column source fields.
- How to use the RDML commands
  - CLR_LIST
  - ADD_ENTRY
  - SELECT / ENDSELECT
  - SUBROUTINE / ENDROUTINE
  - EXECUTE
  - FRM120 Trigger Function

# FRM065 - Using List Components

#### **Objectives:**

- To create a small application, which searches the employee file by name, by start date or by department code.
- To learn how to use the tab folder to display employee details and holidays.
- To show more examples using the SELECT command.

Search Search By Search By By Name By Start Date By Location		Employee	e Surname 🤄	5	Search	
Employee Number	Employee	full na	Department Code	Department Desc	Start date (YYM	T
A1003	Smithe, Ro		FLT	Department Dese	85/12/21	
	Smithers, J		TRVL		81/06/01	
	Smiths, Pe		ADM		71/02/01	
	Smithson,		ADM		89/07/03	
1004	Smithson,	Ruth	ADM		80/05/01	1
1002	Smythe, Jo	hn	ADM		77/01/01	
1009	Snashall, D	amian	AUD		83/12/01	
1008	Sneddon.	Alison	AUD		86/12/01	1
Details Holidays						
Employee Numb	er	A0193				
Employee Surna	me	Smiths	on			
Employee Given		Ruth				
Start date (YYMM	ADD)	89/07/0	13			
Department Cod	e	ADM				
Employee Salary		25,900.	04	÷.		
Notes						_

To achieve these objectives you must complete the following:

- Step 1. Create Form Using Lists
- Step 2. Make Radio Buttons Show and Hide Fields
- Step 3. Add Search Logic
- Step 4. Add Tab Folder and Tab Sheets to the Form
- Step 5. Populate the Tab Sheets

Summary

### **Before You Begin**

You should complete all preceding exercises.

You must also have completed the *Repository* exercise, *REP005* and *REP010*.

#### **Important Note:**

This exercise uses the files **iiiEmployees**, **iiiDepartments** and **iiiEmpHolidays** which you created in repository exercise *REP005* and *REP010*.

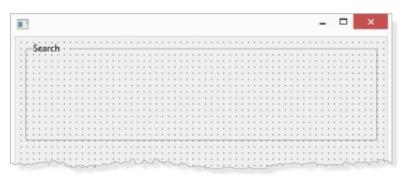
### **Step 1. Create Form – Using Lists**

The form will contain at the top, a "Search" group box, with radio buttons to select "search type". The required search field (e.g. SURNAME) will be displayed based on the radio button selected. A Search button will populate a list view with employees.

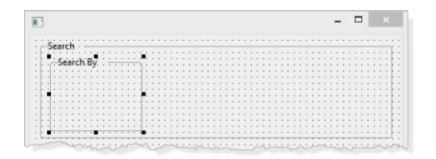
- 1. Start by creating fields in the Repository which will be used for search values.
  - Field iiiSRCDTE, Start Date reference field iiiStartDate
  - Field iiiSRCDEP, Department Code, reference field iiiDeptCode
  - Field iiiSRCNME, Employee Surname, reference field STD_NAME

**Note:** Field iiiSurname has a reference field of STD_NAME. Field iiiSurname can therefore not be used as a reference field itself.

- 2. Create a *New Form / Basic Form: iiiUsingLists Using Lists*. Make the form RDMLX enabled.
- 3. In the *Design* tab, from *Common Controls*, drop a *Group Box* at the top of the form. Change its *Caption* to **Search** and resize it to occupy the full width of the form. Your form should look like the following:



4. Drop another *Group Box* into the Search group box. Change its *Caption* to **Search by** and resize it and place it on the left hand side. Your form should look like the following:



- 5. Drop three Radio Buttons into the Search by group box.
  - a. Set up their properties as shown:

Button	Property	Value
RDBN_1	ButtonChecked	True
	Caption	By Name
RDBN_2	Caption	By Start Date
RDBN_3	Caption	By Location

b. Align your radio buttons using the *Align* dialog on the *Design* ribbon.. Your form should look like the following:

_																												-		2		l		×	
Search · · · · · ·	•						-																												
			0		-	:			0	:			0	1		;				:						:	2				2				2
Search By			-	η.																															
				• •	-			• •			-		-						-				-	•			-	• •			-	• •			-
: D By Name : :				÷						4				1						•				•			•	• •				• •			
		•	-	11	-				-	•	-		-	•	-	•			-	•		•	-			•	-			•	-			•	-
	11		÷.	11	1				1	1	1			1	2	1	1			1				1		1	0	1		1	0	1		1	2
: By Start Date			-	11											-				-				-				-								2
				4.																			-				-				-				
· Bylliocation	• •	•	-	•	-	•		• •		•	-		-	•	-	•		•	-	•		•	-	•		•	-	• •			-	• •	• •		-
			*	11							-	• •			+	•			-	*			-	• •			-	• •			-	• •			۰.
	1		-	11	-				-	-	-		-		-	1			-	-			-	1			-	1	1		-				-
	-		-	20		0			0	2	2		2	0	0		2.2		-	2			0			0	0				2				2
	-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

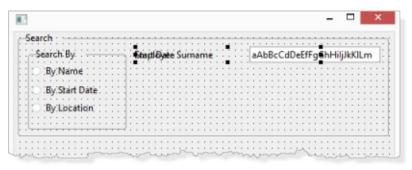
**Note:** *Radio Buttons* are grouped according to the "container" which they belong to, in this case a *Group Box*. Only one radio button can be selected.

Other containers such as *Panel* or the *Form* itself can be used. A *Group Box* is usually used because the interface makes it clear to the user, that the radio buttons are a set.

- 6. Save your form.
- 7. In this step you will place three fields into the Search group box and position them on top of each other. The form will make one visible at any time.
  - a. Drop field iiiSRCNME onto the form. Position the field as shown. Reduce the width of the field. The quickest way to do this is to change the *Width* property to a smaller value and then lengthen the field again if necessary. Try **320**. Resize the form and Search group box if necessary.

																					•	-			1		>	<
Search	 	• •																										
	 11					:	::	:	-		-	1	÷		•								-	• •		•		1.
-Search By	 · Er	mp	loy	ee	Su	m	am	۱e					a	Аb	Bo	:C	dD	)el	ff	g	Gh	Н	ilj.	Jk	KII	m	1	
	 				• •	•		•	-	• •	-	• •	-			-	-			-			-	-	-		-	
By Name	 1.1	11			11	1	1	1	2	11	1			11	1			1		1						1		
	 								-																			
: D By Start Date	 1.1		1.1																									
	 • •	• •			• •	•		•	-	• •	-	• •	•		•		•	-	• -	•	• •	•	-		-	•		•
	 1.1						1.1		0	1	0	1.1			1			2								1		
ByLocation	 								-									-					-					
	 1.1															1.1					• •			• •		1		
	 	• •	• •	• •	• •	•		•	-	• •	*	• •	•		•		•	-	• •		• •	•	-	• •	-			•
	 			_			_		_		_					_		_			_		-	_	_		_	_
	 							•	-		-							-					-					
	 1.1													1.1	1					•				• •				. *

- b. Drop the field iiiSRCDTE on to the Search group box. Change its *ShowTime* property to **false.** Change its *Visible* property to **False.**
- c. Hold down the *Control* key and use the cursor keys to position the file iiiSRCDTE on top of the field iiiSRCNME. Your form should look like the following:



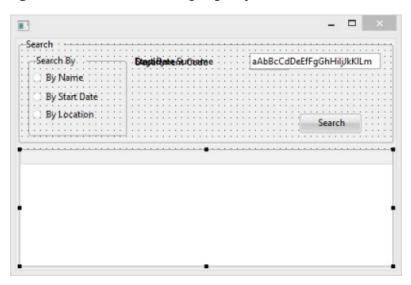
Hint: You can use the *Outlin*e tab to select components.



- d. Drop the field iiiSRCDEP on to the Search group box, and change its *Visible* property to **False. Use the** *Control* + *cursor* **keys to move it on top of field iiiSRCNAME.**
- 8. Drop a *Push Button* on to the Search group box. Change its *Caption* to **Search**, change its *Name* to **SEARCH** and position it as shown.

Search			÷	 	÷		÷		-		-		 ï	-		-	÷				1	1			-		-					-	-
Search	h By: :		•	 i	Đ	eş	ti i i	int	e	ŝ	Ū.	at	 ;	-	: :	;	:	aA	b	Bo	c	d	)el	fF	g	Gh	Hi	ijJ	kК	IL.	m		;
By	Name		;		;		:		-		-		 :	-		-	-		;	-			:		-		-		;			•	•
By	Start D	ate :	:		÷		:		-		-	:	 :	-	: :	-	:		:	-			:		-		-		:				
By	Locatio	ini :	:	 1	:		:					-	 :	-		-	:			•					:		-		:				
	::::		:	 :	:		:	: :	:	::	:	: :	 :	-	::		:		:	-	: :	: :	1		-	Sei	arc	h		ç		-	

- 9. Save your form.
- 10. Make the form longer and drag a *List View* onto the form and resize it as shown. Change the list view *Name* property to **EMPLOYS.**



**Hint:** To position and size the list view:

- a. Drag its handles to resize it to approximately the required size
- b. Hold down the Control key and use the left cursor key to slide it to the left hand side of the form
- c. Hold down the Shift key and use the right cursor key to stretch the list view to the required position at the right hand side of the form.
- 11. From your *Last Opened* tab, expand the file iiiEmployees and add the fields iiiEmployNumber, iiiDeptCode, iiiDeptDescPJF and iiiStartDate to the list view. From *Repository / Fields* add field FULLNAME to the list view.
- 12. On the *Details* tab change the *Display Position* of the FULLNAME column to **2**.

Your form should look like the following:

Search —				
Search By By Name By Start Date By Location	Employee	Surname	aAbBcCdDeEfFgGhHiljJ	kKILm Search
Employee Number	Employee full na	Department Code	e Department Desc	Start date (YYMM
ABCDE	ABCDEFGHUKL	ABCD	aAbBcCdDeEfFg	12/34/56

**Tip:** In a real application you would probably change the column heading *Caption* property for each column and change its *CaptionType* to **Caption**. You could also change the column properties to size and space them as required.

13. Lengthen the form slightly and drop a *Status Bar* onto the form, from the *Controls* tab / *All Controls* view. It will attach to the bottom of the form.

### Step 2. Make Radio Buttons Show and Hide Fields

In this step you will create Click events for the radio buttons to make the required search by field visible.

- 1. Create a Click event for RDBN_1.
- 2. Add logic to make iiiSRCNME visible and hide fields iiiSRCDEP and iiiSRCDTE. Your code should look like the following:

```
* Surname search
Evtroutine Handling(#RDBN_1.Click)
#iiiSRCNME.visible := true
#iiiSRCDEP.visible := false
#iiiSRCDTE.visible := false
Endroutine
```

3. Create Click events for RDBN_2 and RDBN_3. Copy and paste the code from the RDBN_1.Click event into each routine and carefully modify the code as required.

**Tip:** As you have not renamed the radio buttons, it is a good idea to add a comment above each click event defining which search this is.

4. Compile and test your form. Clicking on each radio button should show one field only.

# Step 3. Add Search Logic

In this step you will program the SEARCH.Click event to select from the file iiiEmployees based on the *Search by* radio button settings. The select logic will populate the list view EMPLOYS for each search.

- 1. Create a *Click* Event for the SEARCH button
- 2. Use the Command Assistant to create a Group_By, EMPDATA, containing the fields required to populate list view EMPLOYS. Your code should look like the following:

Group_By Name(#empdata) Fields(#IIIEMPLOYNUMBER #IIIDEPTCODE

Define the Group_by at the form level, below the component definitions. It will later be used in another routine.

**Note:** You could have defined the Group_by in the SEARCH.Click event an itt could still be used in other routines.

- 3. In the SEARCH.Click event routine, complete the SELECT logic to perform if the RDBN_1 is checked. This should search file iiiEmployees on surname, using a logical file, allowing a generic search.
  - a. Add a logical file *iiiVEMP02 By Surname* to your employee file iiiEmployees.
  - b. Give the logical file a key of SURNAME.
  - c. Compile your file. Review your compile options carefully. You need to rebuild *Logical Views* and *OAM*.

Complete your search logic based on the following pseudo code:

Clear list EMPLOYS

If RDBN_1 is checked

Select fields EMPDATA from file iiiVEMP02, with key iiiSRCNME with generic = yesbul

Set up field FULLNAME

Add entry to list EMPLOYS

End select

Endif

Change fields in the Group_by EMPDATA to *default values

Your code should now look like the following:

```
Evtroutine Handling(#SEARCH.Click)
Clr_List Named(#EMPLOYS)
* Surname search
If (#RDBN_1.buttonChecked = true)
Select Fields(#empdata) From_File(iiivemp02) With_Key(#iiiSRCNME) Gen&
#fullname := #iiiSurname + ', ' + #iiiGivenName
Add_Entry To_List(#EMPLOYS)
Endselect
#EMPDATA := *default
Endif
Endroutine
```

- 4. Compile and test your form. Review exercise *REP005* if necessary to check what records were added to file iiiEmployees. In REP005 exercise you created a maintenance form for iiiEmployees. Use this to add more records if required.
- The SELECT which is using GENERIC(*YES) will retrieve all records if the search key is blank.

The list view should now be populated using the Surname search option.

- 5. Create another logical for the employee file iiiEmployees:
  - a. Add a logical file iiiVEMP03 By Start Date.
  - b. Add a key of iiiStartDate.
  - c. Compile the file and rebuild logical files and OAM.
- 6. Extend the SEARCH.Click event to handle the start date search (RDBN_2 checked). The SELECT command has an OPTIONS() parameter which provides additional read options such as read using start key and read backwards. Review the SELECT command in the *Technical Reference Guide* for further details.

Extend your SEARCH.Click logic based on the following

If RDBN_2 is checked

Select fields EMPDATA from file iiiVEMP03, with key iiiSRCDTE using Options, *STARTDATE and *BACKWARDS

Set up field FULLNAME

Add entry to list EMPLOYS

End select

Endif

Your additional code should look like the following:

```
If (#RDBN_2.buttonChecked = true)
Select Fields(#empdata) From_File(iiivemp03) With_Key(#iiiSRCDTE) Optio
#fullname := #iiiSurname + ', ' + #iiiGivenName
Add_Entry To_List(#EMPLOYS)
Endselect
Begincheck
If (#EMPLOYS.items.itemcount = *zeroes)
Message Msgtxt('No Employees from this Start Date')
Set_Error For_Field(#iiiSRCDTE)
Endif
Endcheck
Endif
```

- 7. Compile and test your form. If you added employees with start dates earlier than the current date, enter todays date as your search value. When searching by Start Date, the latest date should be at the top of the list view.
- 8. Extend your SEARCH.Click event to handle the Location search (search by department code iiiSRCDEP). The SELECT command on this occasion should read the logical file iiiEmpByDeptView, with a key of iiiSRCDEP.

Since iiiSRCDEP must exist on the file iiiDepartments (see validation rule at file level, in file iiiEmployees for field iiiDeptCode) your program could check for valid department before continuing the search.

Extend your SEARCH.Click logic based on:

If RDBN_3 is checked

Begin check

Filecheck field iiiSRCDEP in file iiiFILDEPT. Issue a message if not found.

End check

Select fields EMPDATA from file iiiVEMP01, with key iiiSRCDEP.

Set up field FULLNAME

Add entry to list EMPLOYS

End select

Endif

Your additional code should look like the following:

```
* Location Search
If (#RDBN_3.buttonChecked = true)
Begincheck
Filecheck Field(#iiiSRCDEP) Using_File(iiiDepartments) Msgtxt('Department
Endcheck
Select Fields(#empdata) From_File(iiiEmpByDeptView) With_Key(#iiiSRCDI
#fullname := #iiiSurname + ', ' + #iiiGivenName
Add_Entry To_List(#EMPLOYS)
Endselect
Endif
```

9. Compile your form and test it. Select Search by Location, enter an invalid department code and click the Search button.

Based on exercise *FRM035* you should be aware that the FILECHECK will highlight the field iiiSRCDEP and issue a message, if the field does not exist in file iiiDepartments. The ENDCHECK will branch to the ENDROUTINE on error.

Ensure that the list view is populated when a valid department is entered for which employees exist.

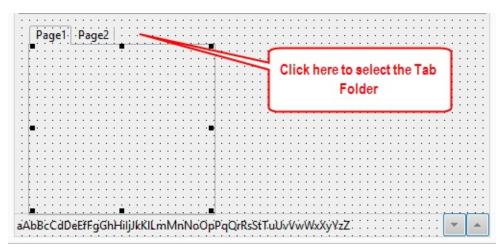
### Sample Solution for SEARCH.Click Event Handling Routine

```
Evtroutine Handling(#SEARCH.Click)
Clr_List Named(#EMPLOYS)
* Surname search
If (#RDBN_1.buttonChecked = true)
Select Fields(#empdata) From_File(iiivemp02) With_Key(#iiiSRCNME) Gene
#fullname := #iiiSurname + ', ' + #iiiGivenName
Add Entry To List(#EMPLOYS)
Endselect
Endif
* Start Date Search
If (#RDBN 2.buttonChecked = true)
Select Fields(#empdata) From File(iiivemp03) With Key(#iiiSRCDTE) Optio
#fullname := #iiiSurname + ', ' + #iiiGivenName
Add Entry To List(#EMPLOYS)
Endselect
Begincheck
If (#EMPLOYS.items.itemcount = *zeroes)
Message Msgtxt('No Employees from this Start Date')
Set Error For Field(#iiiSRCDTE)
Endif
Endcheck
Endif
* Location Search
If (#RDBN 3.buttonChecked = true)
Begincheck
Filecheck Field(#iiiSRCDEP) Using File(iiiDepartments) Msgtxt('Department
Endcheck
Select Fields(#empdata) From File(iiiEmpByDeptView) With Key(#iiiSRCD]
#fullname := #iiiSurname + ', ' + #iiiGivenName
Add Entry To List(#EMPLOYS)
Endselect
Endif
Endroutine
```

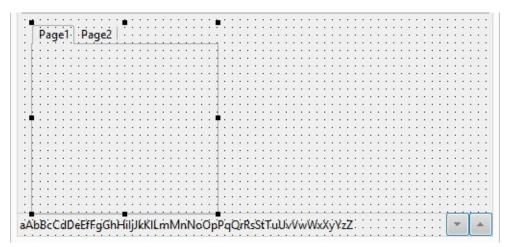
### Step 4. Add Tab Folder and Tab Sheets to the Form

In this step, you will extend the form's functionality by adding a *Tab Folder*, with *Tab Sheets* containing Details and Holidays.

- 1. Extend the length of the form. Drop a *Tab Folder* onto the form below the EMPLOYS list view. Size it to use all the space available.
- **Hint:** The *Tab Folder* automatically contains two *Tab Sheets* and a *Tab Sheet* initially has focus. To enlarge the *Tab Folder* you must select that component. In the *Design* view do this by clicking on the background next to the tabs:



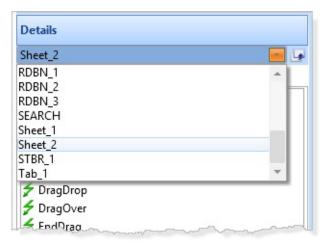
The *Design* view will then look like this:



2. Using the *Details* tab, change tab sheet's (Sheet_1) *Caption* to **Details** and (Sheet_2) *Caption* to **Holidays**. Once again ensure you have selected the correct component. Select a Tab Sheet by clicking on its tab and then click in the middle of the tab sheet area.

Alternatively, you could use the *Outline* tab to select the correct component.

Or, on the *Details* tab, select the component you need from the dropdown list of all comonents in the form:



- 3. Select the application Details tab sheet, by clicking on its tab, and then click in the centre of the tab sheet area.
- 6. On your *Last Opened* tab, expand the file iiiEmployees and select field iiiEmployNumber, iiiDeptCode iiiSurname, iiiGivenName, iiiSALARY, iiiStartDate and iiiEmployNotes. Select the fields by holding down the *Shift* key, click on the first field and then click on the last. Drag these fields onto the top end of the Details tab sheet. Your form should look like the following:

10.0				
Search · · · · · ·				
Search By	PL AD	1.6 8.11.	-ALB-CHD-FET-	Childen and
	Dietach	steñulitatee · · · ·	aAbBcCdDeEfFg	GNHIIJKKILM
By Name 1				
By Start Date	••••••			
: By Location:				
				Search
1				*********
Employee Nu	Employee full	Department C	Department De	Start Date
ABCDE	aAbBcCdDeEf	ABCD	aAbBcCdDeEf	01/01/1900 01:
Details Holidays	•			
Details Holidays Employee Num Department Cor Sumame	ber ABC	D	JkKILm	
Employee Num Department Co	ber ABC de ABC aAb	D BcCdDeEfFgGhHilj		
Employee Num Department Cor Surname Given Name	ber ABC de ABC aAb	D BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj		
Employee Num Department Cor Surname Given Name Salary	ber ABC de ABC aAb 1,23	CD BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj 34,567,890.12		
Employee Num Department Cor Surname Given Name	ber ABC de ABC aAb 1,23	D BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj		
Employee Num Department Cor Surname Given Name Salary	ber ABC de ABC aAb aAb 1,23 01/	CD BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj 34,567,890.12	JkKILm	QrR +
Employee Num Department Cor Surname Given Name Salary	ber ABC de ABC aAb 1,23 01// aAb	CD BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj 14,567,890.12 01/1900 01:00:00	JkKILm 	
Employee Num Department Cor Surname Given Name Salary	ber ABC ABC aAb 1,23 01// aAb SST	CD BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj 14,567,890.12 01/1900 01:00:00 BcCdDeEfFgGhHiljJ	JkKILm v IkKILmMnNoOpPqi bBcCdDeEFFgGhHii	jJk 💠
Employee Num Department Cor Surname Given Name Salary Start Date	ber ABC ABC aAb aAb 1,23 01// aAb sStT KILn	D BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj I4,567,890.12 01/1900 01:00:00 BcCdDeEfFgGhHilj uUvVwWxXyYzZaA	JkKILm kKILmMnNoOpPq bBcCdDeEfFgGhHi tTuUvVwWxXyYzZa	jJk Ab
Employee Num Department Cor Surname Given Name Salary Start Date	ber ABC ABC AAb AAb 1,23 01// aAb 5,27 01// aAb SST KlIn BcC	D BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj 14,567,890.12 01/1900 01:00:00 BcCdDeEfFgGhHiljJ UUvVwWXXyYZZA nMnNoOpPqQrRsS	JkKILm kKILmMnNoOpPq bBcCdDeEfFgGhHi tTuUvVwWxXyYzZa .mMnNoOpPqQrRs	jJk Ab StT
Employee Num Department Cor Surname Given Name Salary Start Date	ber ABC ABC AAb AAb 1,23 01// aAb 5,27 01// aAb SST KlIn BcC	CD BcCdDeEfFgGhHilj BcCdDeEfFgGhHilj 34,567,890.12 01/1900 01:00:00 BcCdDeEfFgGhHiljJ uUvVwWxXyYzZaA nMnNoOpPqQrRs5 dDeEfFgGhHiljJkKIL	JkKILm kKILmMnNoOpPq bBcCdDeEfFgGhHi tTuUvVwWxXyYzZa .mMnNoOpPqQrRs	jJk Ab StT

- 6. Select the Holidays *Tab Sheet*.
- 7. Drop a *Grid* component onto the Holidays *Tab Sheet* and resize it to occupy all of the space. Change the Grid's *Name* to **HOLIDAYS**.
- 8. On the *Last Opened* tab, expand the file iiiEmpHolidays. Drag and drop the fields iiiHOLCDE, iiiHOLSTA, iiiHOLEND, iiiHOLCOM into the HOLIDAYS grid.
- 9. With the Holiday Comments column selected, change its *WidthType* to **Remainder**.
- 10. Save your form

### **Step 5. Populate the Tab Sheets**

In this step you will create an *ItemGotFocus* event routine for the *List View* EMPLOYS, and fetch employee data and select holiday data.

- 1. Add fields iiiSALARY to the Group_by EMPDATA.
- 2. Complete the EMPLOYS.*ItemGotFocus* event routine based on the following:

Fetch fields EMPDATA from the file iiiEmployees with the key iiiEmployNumber

Clear the list HOLIDAYS

Select HOLIDAYS from the file iiiEmpHolidays with the key iiiEmployNumber

Add entry to HOLIDAYS

End select

Your code should look like the following:

Evtroutine Handling(#EMPLOYS.ItemGotFocus) Options(*NOCLEARMESS Fetch Fields(#empdata) From_File(iiiEmployees) With_Key(#IIIEMPLO Clr_List Named(#HOLIDAYS) Select Fields(#HOLIDAYS) From_File(iiiEmpHolidays) With_Key(#IIIEN Add_Entry To_List(#HOLIDAYS) Endselect Endroutine

3. Add code to the end of the SEARCH.Click event to clear the employee fields using the Group_By and clearing the HOLIDAYS Grid. Your code should look like:

. . . . .

#EMPDATA := *default

Clr_List #HOLIDAYS

Endroutine

4. Test your completed application. You should be able to search by one of three methods. Changing focus to an entry in list view EMPLOYS should display employee details and populate the HOLIDAYS grid.

### Summary

### **Important Observations**

- The SELECT/ENDSELECT command is a powerful I/O command which reads multiple records.
- Radio buttons behave as a "set". Only one can be checked at a time.
- The Tab Folder and Tab Sheets enabled a lot of information to be shown on one form

# **Tips & Techniques**

• Review the SELECT command in the *Technical Reference* guide for more information and examples.

# What You Should Know

- How to use the SELECT command
- How to use a simple Tab Folder component
- How to use radio buttons
- How top show and hide fields

# FRM075 - Using a Working List

### **Objectives:**

- To learn how to locate, update and delete entries from a list.
- To introduce the SELECTLIST / ENDSELECT commands for processing lists.
- To show how working lists can be used with the list view.
- To create an application that allows the user to select multiple records from a list to calculate total salaries.

Code	Sumame	Given Name	Salary		
40070	Brown	Veronica	50,125.00		
40090	Bloggs	Fred	20.045.91		
40193	Smithson	Fred	35,000.04		
40907	Simpson	Anne	34,213.04		
1001	Jones	Shirley	2,345.82	E	
1002	Smythe	John	25,000.04		
41003	Smithe	Robert	31,000.04		And the Line Frankler
41004	Smithson	Ruth	21,000.04		Working List Entries
41005	Smiths	Peter	46,700.04		4
41006	Smithers	Jack.	25,000.04		
41007	Snell	George	26,780.04		Compute Selected
41008	Sneddon	Alison	450,000.04		
A1009	Snashall	Damian	31,000.04		Calc. Total Salary
41010	Perry	Kate	60,000.04		
41011	Perron	Christopher	25,000.04		Total Salary
41012	Paul	Patrick.	26,456.04		100.091.81
41013	Pattinson	George	78,977.04		100,031.01
1014	Moore	John	68,000.04		
1015	Woods	Bradley	313,000.04		Total Salary
1016	Tumer	Jack	22 000 04	*	100.091.81
(		111		P	100,091.81

- When the user selects items in the list, the total salary number will be updated.
- To show how 'selected items' in a list component can be selected.
- A working list in the program will also be updated with the selected items.
- When the *Calc. Total Salary* button is pressed, the salary is calculated using the working list. This value matches the total salary. Also, the total items in the working list is displayed.
- To use the TRANSFORM_FILE Built in Function to create a CSV file from the working list.

To achieve these objectives you must complete the following:

Step 1. Select Multiple Entries

Step 2. Use a SELECTLIST Command

Step 3. Build a Working List Step 4. Create a CSV File Summary

# **Before You Begin**

Complete all preceding exercises.

### **Step 1. Select Multiple Entries**

- 1. Create a *New Form /Basic Form***iiiUseWrkList Using a Working List** and make the form RDMLX enabled.
- 2. From the *Controls* tab, drop a *List View* onto the form and resize it. Change the *List View's Name* property to **EMPLOYS**.
- 3. Drop a *Status Bar* onto the form. It will attach to the bottom of the form. Your form should look like the following:

3		x
	·····	
		•
		-
		-
		. *
		1
		1
		-
		1
		-
		-
		T
4bBcCdDeEfFgGhHiljJkKlLmMnNoOpPqQ	In Rest FUUVVWWXXVYZZ	

4. On the *Repository* tab, locate the file PSLMST and expand its definition. Drag and drop the fields EMPNO, SURNAME, GIVENAME and SALARY into the list.

Change the column headings as follows:

Field / column	Property	Value
EMPNO	Caption	Code
	CaptionType	Caption
SURNAME	Caption	Surname
	CaptionType	Caption
GIVENAME	Caption	Given Name
	CaptionType	Caption

# SALARY Caption Salary CaptionType Caption

Resize the fields to display their content. Your *List View* should look like the following:

ABCDE aAbBcCdD aAbBcCdDeEfFg 123,456,789.12	Code	Surname	Given Name	Salary	-	• •		•	-	• •	:		;
2					-			:			:		:
	100000			,,	-		:	:			•	: :	:
					1			:	1				;
					-						:		
					1	1	1	:	1	: :	:	: :	:
					-	• •	•	•	-	• •	•	• •	•
						•			-				

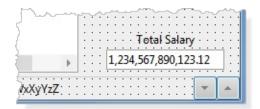
5. Create an *Initialize* event for the list view EMPLOYS.

**Hint:** You can create a click event by selecting EMPLOYS. On the *Details* tab, select the *Events* tab and double click on *Initialize* event. Alternatively, use the right mouse menu on the list view EMPLOYS and use the *Events* option to create an *Initialize* event routine.

6. Add code to the *EMPLOYS.Initialize* event routine, to clear the list EMPLOYS and read all employee records, adding entries to EMPLOYS. Your code should look like the following:

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

- 7. Create a field in the *Repository* called iiiTOTSAL. Define it as **Packed** (15,2) field and give it an edit mask of **J** ( a format of 1,234,567.89-).
- 8. Drop the field iiiTOTSAL onto the bottom right hand side of your form. Change its *LabelPosition* to **Top**.



- 9. Select the EMPLOYS list view. On the *Details* tab, notice that the *List View* has a default *SelectionStyle* of **Multiple**.
- 10. Create an *ItemGotSelection* event routine for list view EMPLOYS. Add code to add SALARY to iiiTOTSAL.
- 11. Create an *ItemLostSelection* event routine for list view EMPLOYS. Add code to subtract SALARY from iiiTOTSAL. Your code should look like the following:

```
Evtroutine Handling(#EMPLOYS.ItemGotSelection) Options(*NOCLEARME
#IIITOTSAL += #salary
```

Endroutine

Evtroutine Handling(#EMPLOYS.ItemLostSelection) Options(*NOCLEARM #IIITOTSAL -= #salary

Endroutine

In an RDML form you could have used:

```
Change Field(#IIITOTSAL) To('#iiitotsal + #salary')
```

12. Compile and test your form. Select multiple entries in the *List View* (hold down the *Control* key). Total Salary should show the correct total for selected employees.

Code	Sumame	Given Name	Salary			
A0070 A0090	Brown Bloggs	Veronica Fred	50,125.00 20,045.91			
A0193	Smithson	Fred	35,000.04			
A0907 A1001 A1002	Simpson Jones Smythe	Anne Shirley John	34,213.04 2,345.82 25,000.04		E	
A1003	Smithe	Robert	31,000.04			
A1004	Smithson	Ruth	21,000.04			
A1005	Smiths	Peter	46,700.04			
A1006	Smithers	Jack	25,000.04			
A1007	Snell	George	26,780.04			
A1008	Sneddon	Alison	450,000.04			
A1009	Snashall	Damian	31,000.04			
A1010	Perry	Kate	60,000.04			
A1011	Perron	Christopher	25,000.04	-		
A1012	Paul	Patrick	26,456.04			
A1013	Pattinson	George	78,977.04			
A1014	Moore	John	68,000.04			
A1015	Woods	Bradley	313,000.04			Total Salary
A1016	Tumer	Jack	22 000 04		+	
•		III				247,457.20

# Step 2. Use a SELECTLIST Command

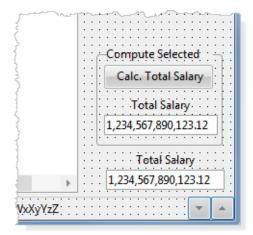
In this step you will add another copy of the iiiTOTSAL field to the form and a push button. Clicking the push button will read through the *List View* EMPLOYS using a SELECTLIST/ENDSELECT command and total the currently selected entries into the new total field.

- 1. Drop a *Group Box* onto the right hand side of your form. Resize it and change its *Caption* property to **Compute Selected**.
- 2. Drag and drop field iiiTOTSAL into the group box. Note that it will be renamed to iiiTOTSAL_1. Changes its *LabelPosition* to **Top**.
- 3. Drag and drop a push button into the group box.

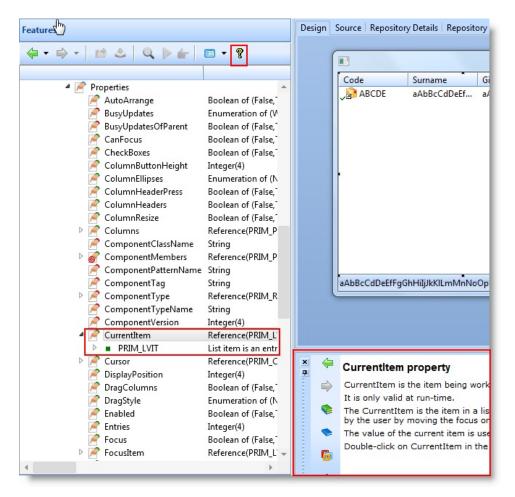
Change its *Name* to **COMPUTE** and *Caption* to **Calc. Total Salary.** 

Create a COMPUTE.Click event routine.

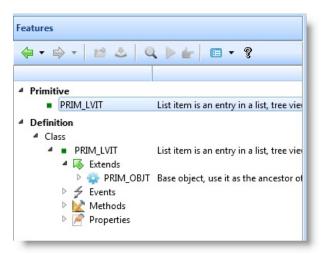
Your form should look like the following:



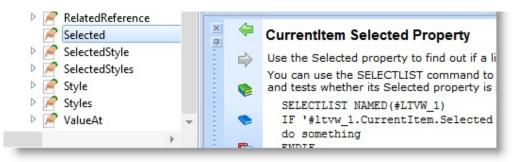
- 4. In this step you will complete the COMPUTE click event logic.
  - a. SELECTLIST/ENDSELECT will read all entries in a list component.
  - b. The selected entries can be identified by using their *CurrentItem* property. Entries in a list are themselves a component. *CurrentItem* has a *Selected* property (**true** or **false**).
  - c. Use F2 Feature help on a component to discover all its events, properties and methods and the help associated with each of these.



d. Double-click PRIM_LVIT to see the events, properties and methods of a list item:



e. Expand the list item properties and choose the Selected property to see its description:



The following code will accumulate SALARY into iiiTOTSAL for selected items:

```
Evtroutine Handling(#COMPUTE.Click)
#iiitotsal_1 := *zeroes
Selectlist
If (#EMPLOYS.currentItem.selected = true)
#IIITOTSAL_1 += #salary
Endif
Endselect
Endroutine
```

Code	Surname	Given Name	Salary		
A0070	Brown	Veronica	50,125.00	*	
A0090	Bloggs	Fred	20,045.91		
A0193	Smithson	Fred	35,000.04		
A0907	Simpson	Anne	34,213.04		
A1001	Jones	Shirley	2,345.82		
A1002	Smythe	John	25,000.04		Compute Selected
A1003	Smithe	Robert	31,000.04		
A1004	Smithson	Ruth	21,000.04		Calc
A1005	Smiths	Peter	46,700.04		Calc
A1006	Smithers	Jack	25,000.04		Total Salary
A1007	Snell	George	26,780.04		50,000.08
A1008	Sneddon	Alison	450,000		
A1009	Snashall	Damian	31,000.04		
A1010	Perry	Kate	60,000.04		Total Salary
A1011	Perron	Christopher	25,000.04		50,000.08
A1012	Paul	Patrick	26,456.04	-	

5. Compile and test your form:

# Step 3. Build a Working List

The DEF_LIST command can be used to define two types of list:

- A browselist (*Type(*browselist)*), used to define an output list in a 5250 RDML function (a subfile in RPG terminology). In web event functions a browselist is used to define output of a list to the web page
- A working list (*Type*(*working)), used to define a list or table in memory. In RPG terminology, a working list would be described as a "multioccurrence data structure". In WAMs, a working list may also be used to define a list to be output the page.

The following defines a working list containing an unlimited number of entries, within platform limits.

Def_List Name(#emplist) Fields(#empno #surname #givename) Type(*Workir

This is the recommended way to define maximum list entries in an RDMLX enabled partition.

- The list will expand as required, and only occupy the space actually required.
- With a fixed number of entries defined, the list will occupy that space in memory, irrespective of the actual number of entries added.

**Note:** By default a working list has an *Entrys* parameter of **50**.

See the *Technical Reference* guide for further details on the *DEF_LIST* command.

In this step you will define a working list and use it to contain currently selected entries in the list view EMPLOYS.

1. Define a work field EMPNOKEY in your form based on field EMPNO. You will use this field to store the employee number in the working list. Your code should look like the following:

Define Field(#empnokey) Reffld(#empno)

- 2. Define a work field LISTCNT based on field STD_NUM. You will use this field to keep a count of the number of entries in the working list.
- 3. Define a working list named EMPLIST, containing fields EMPNOKEY, SURNAME, GIVENAME and SALARY. Set its maximum entries to *max, and use field LISTCNT as a counter. Your code should look like the

following:

Def_List Name(#emplist) Fields(#empnokey #surname #givename #salary) Cc

4. Drag and drop the field STD_NUM onto your form, above the 'Compute Selected' *Group Box*.. Change the following properties:

*LabelPosition* to **Top** 

*LabelHorAlignment* to **Left**.

*Caption* to **Working List Entries** 

LabelType to Caption.

5. The LOC_ENTRY locates an entry in a list, based on a *Where* clause, for example:

```
Loc_Entry In_List(#emplist) Where(#empno = #empnokey)
```

Extend the EMPLOYS *ItemGotSelection* logic to locate an entry in working list EMPLIST. If not found add an entry to working list EMPLIST. Your code should look like the following:

Evtroutine Handling(#EMPLOYS.ItemGotSelection) Options(*NOCLEARMF #IIITOTSAL += #salary Loc_Entry In_List(#emplist) Where(#empnokey = #empno) If_Status Is_Not(*OKAY) #empnokey := #empno Add_Entry To_List(#emplist) Endif Endroutine

6. Extend the EMPLOYS *ItemLostSelection* event routine, to delete an entry from EMPLIST if an entry was located. Your code should like the following:

Evtroutine Handling(#EMPLOYS.ItemLostSelection) Options(*NOCLEARM #IIITOTSAL -= #salary Loc_Entry In_List(#emplist) Where(#empnokey = #empno) If_Status Is(*okay) Dlt_Entry From_List(#emplist) Endif Endroutine 7. Add code to the COMPUTE button click event to set the form field STD_NUM to the working list count, from LISTCNT. Your code should now look like the following:

```
Evtroutine Handling(#COMPUTE.Click)
#iiitotsal := *zeroes
Selectlist
If (#EMPLOYS.currentItem.selected = true)
#IIITOTSAL_1 += #salary
Endif
Endselect
#STD_NUM := #LISTCNT
Endroutine
```

8. Compile and test your form:

Code	Surname	Given Name	Salary	-	
A0070	Brown	Veronica	50,125.00		
A0090	Bloggs	Fred	20,045.91		
A0193	Smithson	Fred	35,000.04		
A0907	Simpson	Anne	34,213.04		
A1001	Jones	Shirley	2,345.82	E	
A1002	Smythe	John	25,000.04		
A1003	Smithe	Robert	31,000.04		Artestice 1 in Eastern
A1004	Smithson	Ruth	21,000.04		Working List Entries
A1005	Smiths	Peter	46,700.04		4
A1006	Smithers	Jack	25,000.04		
A1007	Snell	George	26,780.04		Compute Selected
A1008	Sneddon	Alison	450,000.04		
A1009	Snashall	Damian	31,000.04		Calc. Total Salary
A1010	Perry	Kate	60,000.04		
A1011	Perron	Christopher	25,000.04		Total Salary
A1012	Paul	Patrick	26,456.04		100,091.81
A1013	Pattinson	George	78,977.04		100,031.01
A1014	Moore	John	68,000.04		
A1015	Woods	Bradley	313,000.04		Total Salary
61016	Turner	Jack	22 000 04		100,091.81
•				H.	100,031.01

# Step 4. Create a CSV File

In this step you will use the TRANSFORM_LIST BIF to create a CSV file from the working list.

- 1. Drag and drop a push button onto the form, above the Working List entries field. Change its *Caption* to **Create CSV File** and *Name* to **CREATE**. Create a *Click* event for it.
- 2. The TRANSFORM_LIST BIF can be used to transform a working list into a text file, with a number of file format options. The output file name will usually be defined including a full path to control where it is written.

LANSA has a number of system variables which may be used to provide a path name which will be valid in both a development and production partition. For example this assignment would place the file named EMPLIST.csv into the Visual LANSA partition \object folder.

#STD_QSEL := *part_dir_object + 'EMPLIST.CSV'

Note: The system variable *part_dir_object provides a path ending in "\".

Built-In functions (BIFs) are executed using the USE command. The USE command has the format:

```
USE Builtin() With_args() To_get()
```

A BIF is a called program, and the arguments it receives and returns are defined in the *Repository*. The *Command Assistant* fully supports BIFs, just like any RDML command. Your CREATE Click event logic should look like the following:

Evtroutine Handling(#CREATE.Click)

#STD_QSEL := *part_dir_object + 'EMPLIST.CSV'

Use Builtin(transform_list) With_Args(#emplist #STD_QSel) To_Get(#io\$ If (#io\$sts = OK)

```
Message Msgtxt('File EMPLIST.CSV created in ..\x_' + *partition + '\obje
Endif
```

Endroutine

**Note:** The message text uses a system variable *partition which contains the 3 character partition name.

3. Compile and test your form. Select entries in the list view. The *Create CSV* 

Code	Sumame	Given Name	Salary	-	Create CSV File
A0070	Brown	Veronica	50,125.00		
A0090	Bloggs	Fred	20,045.91		Working List Entries
A0193	Smithson	Fred	35,000.04		Working List Entries
A0907	Simpson	Anne	34,213.04		5
A1001	Jones	Shirley	2,345.82	=	
A1002	Smythe	John	25,000.04		Compute Selected
A1003	Smithe	Robert	31,000.04		
A1004	Smithson	Ruth	21,000.04		Calc. Total Salary
A1005	Smiths	Peter	46,700.04		
A1006	Smithers	Jack	25,000.04		Total Salary
A1007	Snell	George	26,780.04		130,171.85
A1008	Sneddon	Alison	450,000.04		100,111.00
A1009	Snashall	Damian	31,000.04		
A1010	Perry	Kate	60,000.04		Total Salary
A1011	Perron	Christopher	25,000.04		130,171,85
A1012	Paul	Patrick	26,456.04		130,171.03
A1013	Pattinson	George	78,977.04		
A1014	Moore	John	68,000.04		
A1015	Woods	Bradley	313,000.04	1	
41016	Tumer	.Lack	22 000 04	+	
•		m		•	

*File* button will write a CSV file containing the working list's entries:

4. Use Windows Explorer to locate the output file. For example, in a standard Visual LANSA installation:

 $C:\Program Files\LANSA\x_win95\x_lansa\x_dem\object\EMPLIST.CSV$ 

Where DEM is the partition being used for training.

If your PC has MS Office installed, this file is opened in Excel.

X	🚽 19 v 1	[1 - 1] ▼		
F	ile Ho	me Inse	rt Page	Layout For
	× 7	Calibri	* 11	· A A
				100
Pa	ste 🦪	BIU	* 🛄 *	🔕 - <u>A</u> -
Clin	board 🗔		Font	r <u>s</u>
City	board is		Tone	
	A1		· (*	<i>f</i> _x A1002
1	А	В	С	D
1	A1002	Smythe	John	25000.04
2	A1004	Smithson	Ruth	21000.04
3	A1008	Sneddon	Alison	450000
4	A1010	Perry	Kate	60000.04
5	A1012	Paul	Patrick	26456.04
6				

The next step will use the System_Command Built in Function to automatically display the CSV file in Notepad.

5. Add the highlighted code to your *CREATE.Click* event handling routine Evtroutine Handling(#CREATE.Click)

# Define #retcode type(*dec) length(3) decimals(0) #STD_QSEL := *part_dir_object + 'EMPLIST.CSV' Use Builtin(transform_list) With_Args(#emplist #STD_QSel) To_Get(#io\$sts) If (#io\$sts = OK) Message Msgtxt('File EMPLIST.CSV created in ..\x_' + *partition + '\object fo #std_qsel := ('notepad ' + #std_qsel) Use Builtin(system_command) With_Args(X #STD_QSEL) To_Get(#retco Endif Endroutine

This will display the CSV file in Notepad once the TRANSFORM_LIST BIF has successfully completed:

		Format				
100	5 .	Smiths	5 , P	eter	",35000 25000.04 ,46700.0 an",310 0000.04	04

Omitting 'Notepad' from the execute string would load the CSV file in Excel if you have MS Office installed.

### Summary

### **Important Observations**

- In the event-driven program model, when the user selects multiple items in a list, each item is processed individually by a call to the event that is triggered.
- The SELECTLIST command is very similar to the SELECT command. It creates a looping structure to process multiple records from a list.
- The list operations also support the use of IF_STATUS commands.
- Working lists are created using a DEF_LIST command within the program code. The list definition includes the fields in the list and the size (maximum number of entries) of the list.
- The working list counter is updated automatically.
- The TRANSFORM_LIST BIF will produce a text file, with a number of formats options available, from a working list.
- Windows applications can be called using the SYSTEM_COMMAND BIF.

# **Tips & Techniques**

- Allowing users to select multiple items from a list and then performing some action against the resulting selected list of items is a very common requirement in Windows applications.
- Most lists have a property that indicates whether multiple item selection is to be allowed or not. By default, multiple item selection is supported. You should always think about whether or not you want to support multiple item selection and set this property accordingly.

# What You Should Know

- How to use lists with list views.
- How to locate, update and delete entries from a list.
- How to select items from a list using the SELECTLIST command.
- How to use the following list commands:
  - ADD_ENTRY
  - CLR_LIST
  - DEF_LIST
  - DLT_ENTRY

- LOC_ENTRY
- DEFINE
- How to use the TRANSFORM_LIST Built-in function.

## FRM085 - Update from a Grid

### **Objective:**

• To create a small application with search and update for employee details and employee skills.

Search Start Mo	onth	Surname				
January	* Search		Search	h		
D	Full Name	Details				
-	BROWN, VERONICA	Employee Number	A0070		Department Code	INF
	SMYTHE, JOHN	Employee Surname	BROWN		Section Code	DV
1019	DICKENS, CHARLES	Employee Given Name(s)	VERONICA		Employee Salary	50,125.00
1022	THOMPSON, KELLY	Street No and Name	12 Railway S	treat	Start Date (DDMMYY)	280190
			and the owner where the party of the local division of the local d			
		Suburb or Town	Baulkham H		Termination Date (DDM)	(111) 000000
		State and Country	NSW Austra	lia		
		Post / Zip Code	2153			
		Home Phone Number	(02) 9609 46	27		
		Business Phone Number	(02) 9647 27	88		
		Skill Code	Grade	Comment	Date Acquired	
		4GL Programming M			5/03/93	
		Administratn Part 1 P		Met requirement	30/06/96	
		Administratn Part 2 P			1/03/98	
		Advanced Progra P			10/12/95	
		Communications D			4/05/98	
		Computer Science D		Very good result	5/06/90	
		Company Induction P			20/02/90	
		Company Introdu P		Met requirement	5/02/98	
				1.000	Save	Close

- To demonstrate reading a file using SELECT_SQL
- To demonstrate updating a file based on changed entries in a grid

To achieve these objectives you must complete the following:

Step 1. Create Field iiiMONTH and Visual Picklist

Step 2. Create Form – Update from List

- Step 3. Add Search Logic
- Step 4. Display Employee Details and Skills
- Step 5. Update Employee Details
- Step 6. Update Employee Skills

Step 7. Add Drop Down for Skill Code (optional)

Summary

# **Before You Begin**

Complete all earlier FRM exercises before starting this exercise.

# Step 1. Create Field iiiMONTH and Visual Picklist

The application will search the employee file (PSLMST) using SELECT_SQL, using a search on Start Date (STARTDTER) comparing the month value only. In this step you will create a month field and define a static picklist visualization for it. This will be visualized as a combo box which returns a numeric month number.

1. Create a new field, iiiMONTH based on the following:

Field Name	iiiMONTH
Description	Month
Туре	Signed
Length	2
Decimals	0
RDMLX Enabled	No
Open in the Editor	Yes

2. With the new field open in the editor, select the *Visualization* tab. Use the toolbar button to insert a Static Picklist



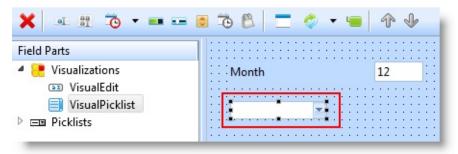
3. In the *Field Parts* panel, select the picklist item and create entries for each month.

🕻 🗉 🕄 🧓 🕶 🎫 🗔	08	♦ - • ■   ↑ ♦	
eld Parts	Image	Caption	Value
📑 Visualizations		January	1
VisualEdit		February	2
VisualPicklist		March	3
Picklists		April	4
Picklist     PKIT 1		May	5
— PKIT_1 — PKIT_2		June	6
- PKIT_3		July	7
— PKIT_4		August	8
— PKIT_5		September	9
— PKIT_6		October	10
— PKIT_7		November	11
— PKIT_8		December	12
— PKIT_9			
— PKIT_10 — PKIT_11			

The default appearance of a picklist is a *DropDownList*. Do not change this setting:

Details			Definition Rules and Triggers Visu
VisualPicklist		•	🗙 🗉 🖞 🐻 🕶 🚥 🖬
E • ?			Field Parts
Properties Events Methods			4 😬 Visualizations
Appearance	DropDownList	-	I VisualEdit
P BusyUpdates	ButtonSet		VisualPicklist
BusyUpdatesOfParent	CheckBox		Picklists
Caption	DropDownList	1	
Columns	Image ImageAndText		
ComponentClassName	ListBox		
ComponentPatternName	EOMMONTH	=	
P ComponentTag			

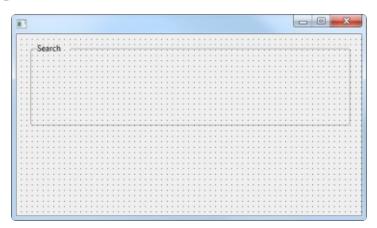
- 4. Set the properties of the *VisualizationPicklist* element in the Details tab:
  - Change its *MarginLeft* to **0**, to hide the label.
  - Change its *DefaultVisual* property to **true**.
  - Resize the DropDownList visualization as shown:



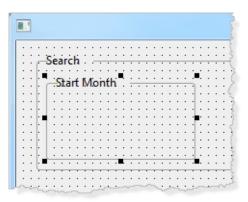
5. Save the field definition and close it.

### Step 2. Create Form – Update from List

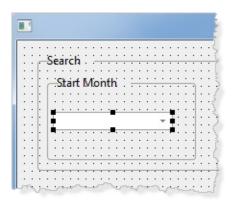
- 1. Create a *New Form / Basic Form* **iiiUpdFromGrid Update from List**. The form should be RDMLX enabled.
- 2. Enlarge the form. Drop a *Group box* to the top of the form. Resize it and change its *Caption* to **Search**. Your form should look like the following:



3. Drop another *Group Box* into the left side of the Search *Group Box* and resize it. Change its *Caption* to **Start Month**. Your form should look like the following:



4. From your *Last Opened* tab, drop field iiiMONTH into the Start Month *Group Box*.



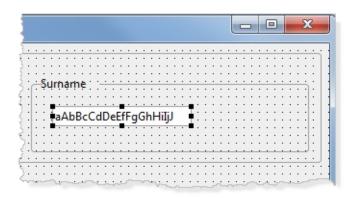
5. From the *Controls* tab, drop a *Push Button* into the Start Month *Group Box*. Resize the *Group Box* as needed and position the *Push Button* as shown.

:	•	•	•				•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
:	:	-	2	è	ir	ċ.	ĥ.	•	٠.	·	•	•						•	•		•					•						•			•	•
		ſ.	~			~	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	Ľ.		•	•	•	•		•		•	•	•	•	•	•	•	÷	· .	•	•	•	•	•	•	•	•	•	•	•	•	•		ŧ.	•	•
•	•	1.1	-	ς.	t:	ir	f	Ň	1 c	١'n	fl		•	÷	•	•	•	-	•	•	•	•	•	•	•	•	•	•	•	•	•	•	_		•	•
	•	ŀ	1.2		~		۰.			~			•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	1		•	•
•	•	ŀ	13	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	1		•	•
	•	ŀ	1.5	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	ŀ	1.	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	1	•	•	•	•	•	•	•	•	•	1	•	•	•	•	•
•	•	1																÷	1	•	1										11	•	1	•	•	•
•	•	ŀ																		•	1											•	•	•	•	•
•	•	ŀ		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	1										з,	•		•	•	•
•	•	ŀ		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
•	•	ŀ		•	•	•	•	·	•	•	•	•	·	•	•	•	•	•	•	•	·	•	•	•	•	•	•	·	•	•	•	•		•	•	·
•	•	ŀ.		•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	-		•	•
•	•	Ŀ		•	•	•	•	•	•	•	•	•	•	•	•	•	•	-	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•
·	•	1	•	•	•	•	•							•			•		•											•						-

- 6. Change the *Push Button Name* to **SEARCH_MONTH** and change its *Caption* to **Search**.
- 7. Save your form.
- 8. From the *Controls* tab, drop a *Group Box* into the Search *Group Box*, on the right hand side. Change its *Caption* property to **Surname**:

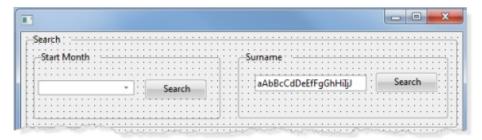
See.																																																				
Des	arç	ņ																																										- 1								1
			-			-		-				 -			-					-				-				-				-				-				-										-		
- 5	ta.	1 1	Mr	mti	h .	-				_		 								_			0																1			1.00	1	1.1	1.1				1.1			
- · · ·						-		-				 -			-	•				-			• •	-	5	110	-	m				-				-				-		-				-				-		
						-						 					•	• •		•	•		• •	F	4	4.	1.4		۴.						,						• •										• •	1.
		4.4		1.1				- 4							-					÷ .				1.4		ъ.												1.14				1.00							1.1			1
100	_				_		_	-	-	1.1	1								٦.	-	÷		• •	-	•	•	• •	-	•		•	-	• •		•	-	• •		•	-	• •	-	•		• •	-			• •	-	• •	
11									Ŧ	1			Se	a	nc.	h							•			•	• •						•		. *		• •			-	• •	1.0										
1-										1.					-								• •	1.		•	• •		•	• •	•	-	• •		•	-	• •		•	-	• •		٠.		• •	-			• •		• •	
1 -	• •		-	• •		-		-	•	• •		 -		•	-	•	-	• •	•	-	1		• •	1	•		• •	-	•		•	-	• •		•	-	• •		•	-	• •				• •	-				-	• •	1.
1.	• •			• •			• •		•					÷.	۰.	۰.	۰.			•	1	5.1		L.		۰.			۰.				1		1	۰.			1	۰.	• •	1.1	1	-			1	-				
11						-						 -			-	•				-				1.		-			•		1	-			÷.	-	•			-		-				-				-		11
·		_							-								-	a	A			•		-	-	-		-	-	_		-	-	-	-	-	-															

9. From the *Repository* tab, *Fields*, drop field iiiSRCNME into the Surname *Group Box*. Change its *MarginLeft* property to **0**, and resize and position it as required.



**Hint:** Temporarily extend both the Search and Surname *Group Boxes* to the right, in order to reduce the width of the field iiiSRCNME. Alternatively, select the field iiiSRCNME and reduce its *Width* property.

10. From the *Controls* tab, drop a *Push Button* into the Surname *Group Box*. Change its *Name* property to **SEARCH_NAME** and change its *Caption* to **Search**. Resize and position it.



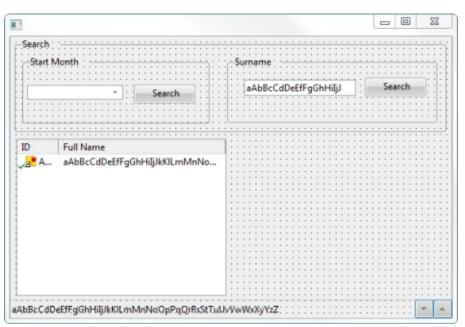
**Hint:** from the *Design* ribbon, the */ Align* dialog has been used to align the Start Month and First Name group boxes at their top edge.

- 11. Save your form.
- 12. Drop a *Status Bar* onto the bottom of the form.
- 13. Drop a *List View* component onto the left side of the form. Resize it. Change its *Name* property to **EMPLOYS**.
- 14. Drop fields EMPNO and FULLNAME into the list view.
- 15. Select the EMPNO column. Change its *Caption* to **ID**. Change its *CaptionType* to **Caption**. Make the column wide enough to display the field EMPNO.
- 16. Select the Full Name column.

Change its *Caption* to **Full Name** and its *CaptionType* to **Caption**.

Change its *WidthType* to **Remainder**.

Your form should look like the following:



17. Save your form.

# Step 3. Add Search Logic

In this step you will create *Click* events for both search buttons and complete the search logic using SELECT_SQL to populate the EMPLOYS *List View*.

- 1. Select the SEARCH_MONTH push button. Change its *Enabled* property to **False** and create a Click event.
- 2. The SQL will have a WHERE clause, which compares STARTDTER with a work field containing a mask '___nn__', where nn is the character value for the month selector field iiiMONTH.

Define a work field (inside the SEARCH_MONTH.Click event) as follows: Define Field(#iiiDATE6) Type(*char) Length(6)

STARTDTER is a signed, length 6 field,

3. The date mask value can be created in field iiiDATE6 using the asString and RightAdjust intrinsic functions.

**Hint:** Use the F2 Feature help on field #iiiMONTH to discover the intrinsic functions (methods) available.

**Note:** On this occasion you are first using the function asString from the numeric field iiiMONTH and then using a RightAdjust function which is available once it is a string. You can use *F2 Feature* help on any Alpha or String field to discover the RightAdjust function. Then use *F1 help* on an intrinsic function (method) to display help text and examples.

The code required is:

#iiiDATE6 := '__' + #iiiMONTH.asDisplayString.RightAdjust( 2, "0" ) + '__'

The **RightAdjust** pads the result to length 2, with a pad character of 0.

Note:

In the next step you will add the SELECT_SQL logic to add rows to list view EMPLOYS.

The SELECT_SQL command is read only and uses SQL instead of calling the LANSA OAM. The OAM uses native I/O on the IBM i server and ODBC on other platforms. SELECT_SQL reads the file directly and is not subject to LANSA validation, virtual field or triggers.

The best way to check your SELECT_SQL logic is to use interactive SQL for the required deployment platform, to ensure that your logic will work as expected with that database (DB/2, MS SQL Server, Oracle?).

There are two forms of SELECT_SQL available.

- In the basic SELECT_SQL command, the SQL logic is defined in the SELECT_SQL WHERE() parameter. This means that there are some restrictions on the type of SQL code which you can implement.
- In the SELECT_SQL Free Format command, the SQL statement is written in the SELECT_SQL USING() parameter. This means that any SQL code which is supported by the database can be implemented in this form of SELECT_SQL. This form of SELECT_SQL can only be used in RDMLX enabled functions and components.

For full information about SELECT_SQL, refer to the *Technical Reference Guide*. You should do a detailed study of this *Technical Reference* information before attempting to use SELECT_SQL.

4. The SQL statement to retrieve employees with a start date in the required month is as follows:

SELECT EMPNO, SURNAME, GIVENAME, STARTDTER FROM XDEMC

Where mm = the month number required.

Complete the SEARCH_MONTH Click event routine as shown below:

```
Evtroutine Handling(#SEARCH_MONTH.Click)
#iiiDATE6 := '__' + #iiiMONTH.asDisplayString.rightAdjust( 2, "0" ) + '__'
Clr_List Named(#EMPLOYS)
#std_strng := 'WHERE (' + '"STARTDTER"' + ' LIKE ' + #quote + #iiiD/
#std_strng := 'SELECT EMPNO, SURNAME, GIVENAME, STARTDTE
Select_Sql Fields(#EMPNO #SURNAME #GIVENAME #startdter) Using
#fullname := #SURNAME + ', ' + #GIVENAME
Add_Entry To_List(#EMPLOYS)
Endselect
#EMPNO #SURNAME #GIVENAME := *default
Endroutine
```

Note: The schema name used for file PSLMST in the SELECT_SQL will

depend on your partition file library name. Visual LANSA creates the SQL table by shortening the library name. For example DCXDEMOLIB, becomes XDEMOLIB as used in the supplied RDML above. Use SQL Management Tools to access the DB and check the schema name being used.

5. Create a Changed event for field iiiMONTH. Add code to this event to enable the SEARCH_MONTH button. Your code should look like the following:

Evtroutine Handling(#IIIMONTH.Changed) Options(*NOCLEARMESSAGE: #SEARCH_MONTH.enabled := true Endroutine

- 6. Compile your form and test it. You should be able to select a month. The search button should populate the list view. **Hint:** Try a number of different months.
- 7. Select the SEARCH_NAME button, change its *Enabled* property to **false** and create a *Click* event for it.
- 8. The SQL statement required to retrieve employees selecting by SURNAME based on in input search value such as 'S%' or %smi%' is as follows:

SELECT EMPNO, SURNAME, GIVENAME FROM XDEMOLIB.PSLMST Where XX = the search criteria.

**Note:** Once again you need to know the schema name for table PSLMST in your local SQL Server database.

Your completed code should look like the following:

```
Evtroutine Handling(#SEARCH_NAME.Click)
Clr_List Named(#EMPLOYS)
#std_strng := #quote + #IIISRCNME + #quote
#std_strng := 'SELECT EMPNO, SURNAME, GIVENAME FROM XDE
Change Field(#EMPNO) To(*blank)
Select_Sql Fields(#EMPNO #SURNAME #GIVENAME) Using(#std_strng
#fullname := #SURNAME + ', ' + #GIVENAME
Add_Entry To_List(#EMPLOYS)
Endselect
#EMPNO #SURNAME #GIVENAME := *blanks
Endroutine
```

9. Create a *Changed* event for the field iiiSRCNME. Complete this event routine to enabled the SEARCH_NAME button. Your code should look like the following:

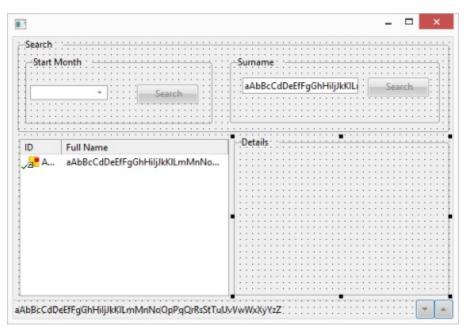
Evtroutine Handling(#IIISRCNME.Changed) Options(*NOCLEARMESSAGI #SEARCH_NAME.enabled := true Endroutine

10. Compile your form and test it. Check the SEARCH_NAME logic using values such as SM% and %Y%.

## Step 4. Display Employee Details and Skills

In this step you will extend the form design and add logic to retrieve employee details and skills when selected in the employees list view.

1. Enlarge the form and drag and drop a Group Box onto the right hand side. Resize the *Group Box* and change its *Caption* to **Details**. Your form should look like the following:



- 2. Expand the file definition on the Repository tab for file PSLMST. Select all the fields from EMPNO to PHONEBUS while holding down the *Shift* key and drag them to the top right side of the Details *Group Box*.
- 3. Reduce the field widths and if necessary widen the *Form* and *Group Box*, and then select fields from DEPTMENT to TERMDATE and drag and drop them to the right of the first set of fields:

Employee Surname	aAbBcCdDeEfFgGhHi		Section Code		AB
Employee Given Name(s)	aAbBcCdDeEfFgGhHi		Employee Salary		123,456,789.12
Street No and Name	aAbBcCdDeEfFgGhHi	aAbBcCdDeEfFgGhHiljJkKILm		VIYY): : : :	123456
Suburb or Town	aAbBcCdDeEfFgGhHi	ljJkKlLm	Termination Date	e (DDMMY	Y) :: 123456
State and Country	aAbBcCdDeEfFgGhHi	ljJkKlLm			
Post / Zip Code	123456				
Home Phone Number	ABCDEFGHUKLMNO				
Business Phone Number	ABCDEFGHUKLMNO				

- 4. Save your form.
- 5. Make adjustments to field width and reduce the width of the Group Box and form if necessary.

Do not spend time on this step. If you follow the Windows development training path, you will learn how to use *Layout Managers* to control your form design in an intelligent way.

- 6. Drop a *Grid* control onto the lower area of the Details group box and resize it. Change its *Name* to **SKILLS**.
- 7. In the *Repository* tab, find the file PSLSKL and expand its definition. Drag and drop field EMPNO into the SKILLS grid. Change its *Visible* property to **false**.
- 8. Drag fields SKILCODE, GRADE, COMMENT and DATEACQ into grid SKILLS. Adjust the *Caption, CaptionAlign* and *CaptionType* so that your *Grid* columns look like the following:

ABCDEFGHIJ A aAbBcCdDeEfFgGh 12/34/56	

9. On the *Repository* tab, expand the file SKLTAB and drop the field SKILDESC into the SKILLS grid. Change its *DisplayPosition* to **2**.

Your grid should look like the following:

Skill Code	Skill Full Description	Grade	Comment	Date Acquired
ABCDEFGHU	aAbBcCdDeEfFgGh A		aAbBcCdDeEfFgGh	12/34/56

10. Add field DATEACQR from the file PSLSKL to the *Grid*. Make the column *Visible* **false**.

The field DATEACQR will be used to recognize entries which were loaded from the file PSLSKL.

- 11. Save your form definition.
- 12. Create an *ItemGotSelection* event for the list view EMPLOYS.
  - a. Define a GROUP_BY for all employee fields on the form. Your code should look like the following:

Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME #ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #PHONEHME #PHONEBUS #DEPTMENT #SECTION #SALARY #STARTDTE #TERMDATE)

b. This event should:

Fetch all fields for current employee number using GROUP_BY EMPDATA

Retrieve all skills (file PSLSKL) for this employee number

Fetch the skill description from file SKLTAB for this skill code

Add entry to grid SKILLS.

Your code should look like the following:

Evtroutine Handling(#EMPLOYS.ItemGotSelection) Options(*NOCLEARMI Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME #ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #PHONEHME #PHONEBUS #DEPTMENT #SECTION #SALARY #STARTDTE #TERMDATE) Fetch Fields(#EMPDATA) From_File(pslmst) With_Key(#EMPNO) Clr_List Named(#SKILLS) Select Fields(#SKILLS) From_File(pslskl) With_Key(#EMPNO) Fetch Fields(#skildesc) From_File(SKLTAB) With_Key(#skilcode) Add_Entry To_List(#SKILLS) Endselect Endroutine

13. Compile and test your form. Complete a search and select an employee in the list view. Your form should look like the following:

earch Start Month					
Statt Month	Sumane				
January 💌 Search		Search	]		
D Full Name	Details				
0070 Brown, Veronica	Employee Number	A1019		Department Code	LEG
1002 Smythe, John	Employee Sumame	Dickens		Section Code	01
1019 Dickens, Charles	Employee Given Name(s)			Employee Salary	45,000.04
A1022 Thompson, Kelly				Start Date (DDMMYY)	01018
	Street No and Name	17 Grantham Ro	ad,		
	Suburb or Town	Seven Hills		Termination Date (DDMMYY	) 000000
	State and Country	NSW/			
	Post / Zip Code	2147			
	Home Phone Number	718 1891			
	Business Phone Number	565 2341			
	Bushess Phone Number	360 2341			
	Skill Code	Skill Full Description	Grade	Comment	Date Acquired
	ADMIN1	Administratn Part 1	D	Met requirement	25/03/98
	COM	Communications Degre	D		4/05/98
		Company Introduction	P		5/02/98
	KEY	Keyboard Skills	P	Met requirement	5/02/98
		Management Course 1		Met requirement	15/03/98
	MANAGE3	Management Course 3	F		2/05/98
	MARKET1	Marketing Course 1	D	Met requirement	25/03/98
	MARKET3	Marketing Course 3	P		1/05/98
	REL	Relation Theory	P		5/05/98
	SHORT	Take Shorthand	P		3/05/98

## **Step 5. Update Employee Details**

In this step you will add *Save* and *Close* push buttons. You will code the *Save* button *Click* event to update the employee record (PSLMST) and update, delete and add skills to file PSLSKL from changes made to the SKILLS grid.

1. Add two push buttons, inside the Details group box, below the SKILLS grid. Set up the buttons as follows:

Save	Caption	Save
	Name	SAVE
Close	Caption	Close
	Name	CLOSE

Your form should look like the following:

ade	Comment	Date Acquired
	aAbBcCdDeEfFgGh	12/34/56
i i i i i i i i i i i i i i i i i i i		
2		
	Save	Close
	Save	Close

2. Create a click event for the CLOSE button and add code to close the form. Your code should look like the following:

Evtroutine Handling(#CLOSE.Click) #com_owner.closeForm Endroutine

3. In the Details *Group Box*, select the field EMPNO and change its *ReadOnly* 

property to **true**.

4. Create a click event for the SAVE push button. Add code to update all fields in the file PSLMST. Check the I/O status and issue a message the status is not OK. Your code could look like the following

```
Evtroutine Handling(#SAVE.Click)
Update Fields(*all) In_File(pslmst) Val_Error(*next)
If_Status Is(*OKAY)
Else
Message Msgtxt('Error occured on Employee update')
Endif
Endroutine
```

**Note:** The UPDATE statement has no *With_Key()* parameter. It is updating the 'last record read'. The LANSA cross update check will be applied, to prevent an update if the record changed since it was read.

5. Compile and test your form. You should be able to update an employee. Validation errors will highlight fields and display messages in the *Status Bar*.

# **Step 6. Update Employee Skills**

In this step you will extend the SAVE *Click* event routine:

- To process all entries in the SKILLS grid (SELECTLIST/ENDSELECT) and update, delete or insert a new skill.
- The SKILLS grid will now be loaded with 5 blanks entries, which may be used to add a new skill.
- The field DATEACQR column will be used to identify "existing" skills entries.
- If the Date Acquired (DATEACQ) is zero, the skill will be deleted.
- All other skill entries will be updated.
- The SKILLS grid will be rebuilt after a successful update to show current entries.
- 1. In the SKILLS grid, select the column Skill Code by clicking on its column heading. Change its *ReadOnly* property to **false**. Repeat this change for columns Grade, Comment and Date Acquired.
- 2. Create a "SKILLS" subroutine:

Move the clear and populate SKILLS logic from the EMPLOYS *ItemGotSelection* routine into the SKILLS subroutine.

Add an EXECUTE SKILLS command to the EMPLOYS *ItemGotSelection* routine

Define a group_by SKLLIST for all the SKILLS grid fields, excluding EMPNO

Set the SKLLIST group_by fields to default values after populating the SKILLS grid

Extend the SKILL subroutine logic to add 5 blank entries to the end of the SKILLS grid

Your SKILLS subroutine should look like the following:

Subroutine Name(SKILLS)Group_By Name(#skllist) Fields(#SKILCODE Clr_List Named(#SKILLS)

Select Fields(*all) From_File(pslskl) With_Key(#EMPNO) Fetch Fields(#skildesc) From_File(skltab) With_Key(#skilcode) Add_Entry To_List(#SKILLS) Endselect #skllist := *default Begin_Loop To(5) Add_Entry To_List(#SKILLS) End_Loop Endroutine

- 3. Compile your form and test it. The SKILLS grid columns should be input capable (except Skill Description) and there should be 5 "blank" entries at the end of the grid. The *Save* button will not yet update employee skills.
- 4. In this step you will extend the SAVE logic to update or delete employee skills (file PSLSKL).

If the employee update is OK

Read all entries in the SKILLS grid using SELECTLIST

If field DATEACQR is not zero

If field DATEACQ is not zero

UPDATE skill with a key of employee number and skill code

Otherwise, delete skill with a key of employee number and skill code

If I/O status is not OK

Update current entry in SKILLS grid

Leave SELECTLIST

Your SAVE logic should now look like the following:

Evtroutine Handling(#SAVE.Click) Update Fields(*all) In_File(pslmst) Val_Error(*next) If_Status Is(*okay) Selectlist Named(#SKILLS) If (#dateacqr *NE *zero) If (#dateacg *NE *zeroes) Update Fields(#SKILLS) In_File(pslskl) With_Key(#EMPNO #skilcode) V Else Delete From_File(pslskl) with_Key(#empno #skilcode) Val_Error(*next) Endif If_Status Is_Not(*okay) Upd_Entry In_List(#SKILLS) leave Endif Endif Endselect If_Status Is(*okay) **Execute Subroutine(SKILLS)** Endif Else Message Msgtxt('Error occurred on Employee update') Endif Endroutine

5. Compile and test your form. You should now be able to update existing skills (e.g. change grade, comment or data acquired) or delete a skill by setting Date Acquired to zero.

Notice that, when a validation error occurs (e.g. invalid Grade), the error is highlighted and processing of the SKILLS grid stops.

6. In this step, you will extend the SELECTLIST logic:

Insert an employee skill record, if DATEACQR is zero and skill code (SKILCODE) is not blank.

If the insert is not OK, issue a message, update the current SKILL entry and stop processing SKILLS entries (using a LEAVE command).

If no errors occurred, execute the SKILLS subroutine to rebuild the SKILLS grid.

Define a Group_By, which contains only the fields needed to insert an employee skill record:

Group_By Name(#skilladd) Fields(#empno #skilcode #dateacq #comment #grade)

Extend your save logic as outlined above. Use the SKILLADD Group_by when inserting an employee skills record.

Your SAVE logic should now look like the following. Changes are shown in red.

```
Evtroutine Handling(#SAVE.Click)
Update Fields(*all) In_File(pslmst) Val_Error(*next)
If_Status Is(*okay)
Selectlist Named(#SKILLS)
If (#dateacqr *NE *zero)
If (#dateacq *NE *zeroes)
Update Fields(#SKILLS) In_File(pslskl) With_Key(#EMPNO #skilcode) Val_
Else
Delete From_File(pslskl) With_Key(#empno #skilcode) Val_Error(*next)
Endif
If_Status Is_Not(*okay)
Upd_Entry In_List(#SKILLS)
Leave
Endif
```

Else

```
If (#skilcode *NE *blank)
Insert Fields(#SKILLADD) To_File(pslskl) Val_Error(*next)
If_Status Is_Not(*OKAY)
Message Msgtxt('New Skill not inserted')
Upd_Entry In_List(#SKILLS)
Leave
Endif
Endif
```

```
Endif
Endselect
If_Status Is(*okay)
Execute Subroutine(SKILLS)
Endif
Else
```

Message Msgtxt('Error occurred on Employee update') Endif Endroutine

7. Compile and test your form. You should be able to add new skill entries and also handle a validation error when inserting a skill.

**Hint:** At this stage, you must know valid skill codes. Delete a skill (Date Acquired = zero) noting the skill code and then re-add the same skill.

Notice that if a validation error occurs on an insert, the error is highlighted and processing of the SKILLS grid stops.

- 8. In this step you will refine the error handling. Test your form as follows:
  - a. Select an employee to display employee details and skills
  - b. Save with invalid data (e.g. surname=blank, salary=zero)
  - c. Select a different employee in the EMPLOYS list view
  - d. Notice that the error fields are still highlighted.

Notice that the EMPLOYS *ItemGotSelection* routine definition has an OPTIONS() setting which does not clear errors and messages:

Evtroutine Handling(#EMPLOYS.ItemGotSelection) **Options(*noclearerrors** 

- 9. Remove the *Options()* parameter from the EMPLOYS *ItemGotSelection* event routine statement.
- 10. Compile and test your form and repeat the test in 8 above. Note that field highlighting is now cleared when a new employee is selected in the EMPLOYS list view.

# Step 7. Add Drop Down for Skill Code (optional)

From LANSA V12 is a new field visualization option: *Dynamic Picklist*. This enables a simple Reusable Part component to be created that populates a picklist from a table file such the skills table (SKLTAB). This reusable part can then be attached to the dynamic picklist visualization for the skill code, such as a combo box.

This step simply illustrates a Dynamic Picklist as a solution. See the *Visual LANSA Developer Guide* for full details and examples.

# What is a Reusable Part?

A reusable part is a component which either extends PRIM_PANL, meaning it is a visual panel or extends PRIM_OBJT, meaning it is a hidden component which is invoked to 'do something'.

1. Create a *New / Reusable Part / Object* called iiiVIS01 – Skills Picklist. Paste the following code into it and compile it:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_OBJT *implements #Prim_dc.iDynam
Mthroutine Name(Load) Options(*redefine)
#Picklist.RemoveAll
Select Fields(*all) From_File(skltab)
#Picklist.Add( #skilcode #skildesc )
Endselect
Endroutine
End_Com
```

#### Note:

- *implements Prim_DC.iDynamicPicklist
- **iDynamicPicklist** allows a visualization reusable to load the picklist at runtime.
- Load method is executed during initialization and any time a monitored value or context changes. This picklist instance is received via the Picklist map. Your method routine Load Options(*Redefine) will contain:

Define_Map For(*input) Class(#PRIM_PKLT) Name(#Picklist) Pass(*BY_]

• The instance of the picklist is maintained at runtime meaning that you

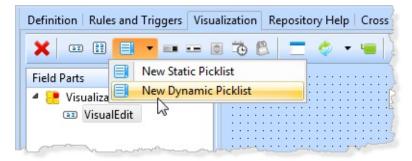
must clear the list if new data is required.

Copy the field SKILCODE to create a new field iiiSKILCODE.
 a.Copy rules and triggers and Help text.

b.Open the new field in the editor.

c.Select the *Visualization* tab.

d.Insert a *New Dynamic Picklist* 



3. Select your reusable part (iiiVIS001) in the *Repository Find* dialog and click OK.

Repository Find		
🗇 Reusable Part		
Name	Description	
Þ 🧔 A		
🖻 🧔 В		
Þ 🧔 C		
Þ 🧔 D		
4 🗇 E		
EOEXAM05	Sample RP	
EOEXAM04	Sample RP	
EOEXAM21	Button B	
COMVIS01	Skills list	
EOEXAM20	Button RP	
Þ 🜍 F		
G		
▷ 🧔 H ▷ 📩 T		
Þ 💩 K		
⊭ 🦦 K ▷ 📩 I		
Þ 🙆 M		
⊭ 🥪 M ⊳ 🙈 N		
		OK <u>C</u> ancel

4. Select the *VisualPicklist* in the *FieldParts* list. Lengthen the combo box (which will display Skill Description) and change its *DefaultVisual* property to **true**.

Definition   Rules and Triggers   Visi	ualization Repository Help Cro	ss References
X = # * • = =	🖲 to 🗳 📃 🚍 🛷 🕶	I I I I I I I I I I I I I I I I I I I
Field Parts		
Visualizations	Skill Code	ABCDEFGHU
📼 VisualEdit		
VisualPicklist	Skill Code	
▲ □ Picklists		••
📼 DynamicPicklist		

- 5. Save and close your field definition.
- 6. You will now change the SKILLS grid second column to use iiiSKILCODE, and add code to set up its value and use the correct value when updating the Personnel Skills file from the SKILLS grid.
  - a. If necessary open form **iiiUpdFromGrid** in the editor.
  - b. Change the SKILLS grid column 2 to use iiiSKILCODE.

```
Define_Com Class(#PRIM_GDCL) Name(#GDCL_2) Displayposition(1)
Parent(#SKILLS) Readonly(False) Source(#iiiSKILCODE)
```

c. In the SKILLS subroutine, set up field iiiSKILCODE before adding an entry to grid SKILLS.

• • • •

#iiiskilcode := #skilcode

Add_Entry To_List(#SKILLS)

• • • •

d. In the SAVE.Click routine, set up field SKILCODE after each entry in the SKILLS list is read:

... Selectlist Named(#SKILLS)

#skilcode := #iiiskilcode

•••

7. Select the *Design* tab. Select the column heading for the Skill Code column in the SKILLS grid and change its *UsePicklist* property to **true**.

Right click on the Skill Description column heading and use the context menu to *Delete Component*. This column is no longer necessary.

Skill Code	Grade	Comment	Date Acquired	
DEFGHU	Δ.	aAbBcCdDeEfFgGh	12/34/56	

8. Compile and test your form. The Skill Code column (which is ReadOnly **false**) displays the current skill description for each row.

Skill Code	Grade	Comment	Date Acquired
Administratn Part 1	D	Met requirement	25/03/98
Communications Degre	D		4/05/98
Company Introduction	P	Met requirement	5/02/98
Keyboard Skills	P	Met requirement	5/02/98
Management Course 1	D	Met requirement	15/03/98
Management Course 3	F		2/05/98
Marketing Course 1	D	Met requirement	25/03/98
Marketing Course 3	Р		1/05/98

9. Click on any cell in the Skill Code column to display a combo box containing all entries from the Skills Table (SKLTAB):

Skill Code	Grade	Comment
Administratn Part 1	D	Met requirement
cations Degre 📃 💌	d D	
4GL Programming 🔒	M	Met requirement
Accountancy Degr	P	Met requirement
Administratn Part 1 – Administratn Part 2	М	
Advanced Program	D	Met requirement
CL Programming	P	
Communications De	P	
Computer Science 📑		

You can select a new value to change the employee skill. However, your save logic expects skill code cannot be changed (employee skill file is keyed on employee number and skill code).

- 10. In this step you will control whether the skill code column is read only. For existing skill entries it should be *ReadOnly* = **True**. For the five blank entries, skill code column should be *ReadOnly* = **False** 
  - a. Change the SKILLS subroutine, To set column 2 ReadOnly= True, using the following code, after each entry is added:

```
Add_Entry To_List(#SKILLS)
```

#skills.cell<#skills.currentitem.entry 2>.readonly := true

• • • •

As always, *F2 Features* help will enable you to find the information you need about the Grid component.

The cell property is used like this:

#skills.cell<row column>

b. Change iiiSKILCODE to *blanks, before five blank entries are added. The skills combo box will now show blanks initially.

... #skllist := *default

#iiiskilcode := *blanks

Begin_Loop To(5)

•••

11. Compile and test your form. For an employee with existing skills you change grade, comment and date acquired only. Use one of the five blank rows to add a new skill. The skill code combo box visualization allows a valid skill to be selected.

## Summary

#### **Important Observations**

- This application is a simple form which is designed to extend your RDML/RDMLX knowledge. It is not intended to represent the best user interface for this type of "header and detail" update.
- SELECT_SQL can be used to create powerful and flexible search logic for reading files.
- Field visualizations can be defined easily and rapidly and are then available in all Windows applications.
- Group boxes enable related data and objects to be visually grouped on the form
- A grid component can have input capable columns and may be processed to handle update, delete and insert to the related file.

# **Tips & Techniques**

- Always use interactive SQL to test your proposed logic over your database platform.
- To use SELECT_SQL effectively you need a good knowledge of SQL.
- To highlight validation errors in a grid, use the UPD_ENTRY command.

# What You Should Know

- How to use SELECT_SQL
- How to write a simple "header/detail" update form.
- How to manipulate a cell in a grid (specific row and column).

# FRM095 - Calling a Function

In this exercise you will create a simple form (Salary Review) which calls a function passing department code and a number which represents "percentage salary increase". The called function creates a list of employees for the requested department, containing both current and new salaries based on the percentage increase. This results list is returned to the calling form. The Salary Review form displays the results in a list view and calculates the total cost of the increase.

The called function will be compiled for Windows and will also be checked in and compiled on the IBM i server.

The "Salary Review" form will be tested in two different modes:

- As a Windows Application running on the desktop, with a local database.
- As a client to an RDMLX IBM i Server

The form will use a system variable *SSERVER_CONNECTED (value Y or N) to determine whether to CALL the local function, or USE

CALL_SERVER_FUNCTION to call the function running on the IBM i server.

Request					
Department Code		ADM		Submit	
Percent Increase		4.00			
Employee Num	Employee full n	Department Co	Employee Salary	New Salary	
A1001	JONES, BEN	ADM	2,345.82	2,439.65	
A1012	PAUL, PATRI	ADM	26,456.04	27,514.28	
A1013	PATTISON, G	ADM	78,977.04	82,136.12	Ξ
A1015	WOODS, BR	ADM	313,000.04	325,520.04	
A1020	DOUGLAS, A	=	121,500.04	126,360.04	
A1021	MCCULLY, D	ADM	87,000.04	90,480.04	
A1025	ROBINSON,	ADM	44,455.04	46,233.24	
A1027	MORRISON,	ADM	1,878,773.04	1,953,923.96	
A1111	VEREY, WAR	ADM	45,678.04	47,505.16	
A1404	MRS BRICK,	ADM	12,345.04	12,838.84	-
		Total Sa	lary 113,4	49.59	

## **Objective:**

• To demonstrate the EXCHANGE and CALL commands and the CALL_SERVER FUNCTION BIF.

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

Step 1. Create the Called Function

Step 2. Create Salary Review FormStep 3. Test Salary Review ApplicationSummary

# **Before You Begin**

Complete all earlier FRM exercises.

This exercise requires access to an IBM i server.

## **Step 1. Create the Called Function**

- Create the following new fields:

   a.iiiNEWSAL, New Salary, Packed, 11, 2 Edit Mask N
   b.iiiPERCNT Percent Increase, Packed, 3, 2, Edit Mask N
   c.iiiTOTSAL, should already exist. It was created in exercise FRM075.
- 2. Check the new fields into the IBM i server.
- 3. Create a new process **iiiPRO01 FRM Training**. You do not need to open it in the editor.
- Create a new function iiiFN21 Salary Review, belonging to process iiiPRO01. Create the function RDMLX enabled.
- 5. Functions which receive and return a working list require a FUNCTION statement with a RCV_LIST() parameter. Define your FUNCTION statement as follows:

Function Rcv_List(#Employs)

- 6. Define a working list EMPLOYS containing fields EMPNO, FULLNAME, DEPTMENT, SALARY and iiiNEWSAL. Entrys=*max
- 7. Define a GROUP_BY, EMPDATA, containing fields EMPNO, SURNAME, GIVENAME, DEPTMENT, SALARY.
- 8. Your function logic should be based on the following:

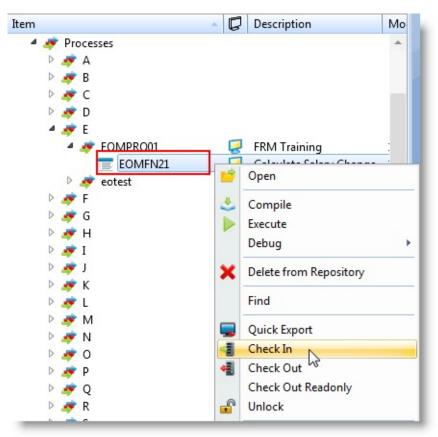
Clear list EMPLOYS SELECT from file PSLMST1, using group_by EMPDATA, with a key of DEPTMENT Assign Fullname to Surname + Givename. Calculate iiiNEWSAL as (SALARY + (SALARY * (iiiPERCNT / 100)) Add entry to EMPLOYS End selection Return

9. Write your RDMLX based on the above. If necessary refer to Appendix A.

FRM095 for a solution.

- 10. Compile your function
- 11. Check in and compile the function to the IBM i Server.

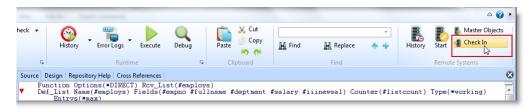
Select your function IIIFN21 on the *Repository* tab, or *Last Opened* tab and use the right mouse menu to Check In.



The process iiiPRO02 will be automatically checked in and compiled as well.

**Note:** Always save your source or compile it locally (this automatically saves the current source) when using check in from tabs such as *Repository* or *Last Opened*.

Alternatively, check in the object currently open in the Editor using the Check In command in the ribbon:



You will learn more about Check In in exercise FRM125 - Check Out / In to IBM i.

## **Step 2. Create Salary Review Form**

- 1. Create a new form **iiiCOM17 Salary Review Request**. Switch to the Design tab.
- 2. Create the form as follows:
  - Drop a Group Box towards the top of the form and resize it. Give it a *Caption* of **Change Request**.
  - Drop fields DEPTMENT and iiiPERCNT onto the Group Box and position them as required. Remember you can use the *Edit / Align* dialog to align a group of components.
  - Drop a Push Button into the Change Request group box, to the right of the fields. Change *Name* to **SUBMIT**. Give it a *Caption* of **Submit** and create a Click event routine for it.
  - Drop a List View onto the form and resize and position it. Add fields EMPNO, FULLNAME, DEPTMENT, SALARY and iiiNEWSAL.
  - Add a Status Bar to the form.
  - Add field iiiTOTSAL to the form below the List View.

Your form should look like the following:

	]						x
-	-Request						
	Department	Code	ABCD		Sub	mit	
i	Percent Incr	ease	1.12				
:							21
	Employee Nu	Employee full .	Department	C Emp	loyee Sal	New Salary	:
	ABCDE	ABCDEFGHIJ	ABCD	123,	456,789.12	123,456,789.1	.2
	· · · · · · · · · · · · · · · · · · ·		Total	Salary	1,234,56	7,890,123.12	
aAbBcCdDeEfFgGhHiIjJkKILmMnNoOpPqQrRsStTuUvVwWxXyYzZ							

- 3. Switch to your program Source. Define four work fields:
  - NEWTOT, reference field iiiTOTSAL
  - CURTOT, reference field iiiTOTSAL
  - SRVCON, *CHAR, 1, default = *SSERVER_CONNECTED
  - SSTATUS, reference field IO\$STS
- 4. Define the EMPLOYS working list. This must be identical to the DEF_LIST definition in function iiiFN21.

Def_List Name(#employs) Fields(#empno #fullname #DEPTMENT #salary #i

It is a good idea to copy the DEF_LIST definition from function iiiFN21 as it is essential that your working lists are identically defined in the calling form and the called function. If not identical, run time errors will occur. Each list must contain the same fields in the same sequence.

5. Review the following pseudo code. You will create this in the following steps.

Use BEGINCHECK/FILECHECK/ENDCHECK to validate DEPTMENT using file DEPTAB

Clear list EMPLOYS

EXCHANGE fields DEPTMENT and iiiPERCNT

If SVRCON = N

Call, process *direct, function iiiFN21, Pass list EMPLOYS

else

USE CALL_SERVER_FUNCTION, with arguments (*sserver_ssn iiiFN21 Y *default EMPLOYS) To Get(#sstatus)

EndIf

If (#sstatus *ne OK)

Message 'Call server function failed'

End if

Clear list LTVW_1

CURTOT and NEWTOT = zero

Selectlist EMPLOYS

Accumulate SALARY in CURTOT

Accumulate NEWSAL in NEWTOT

Add entry LTVW_1

End selectlist

iiiTOTSAL = NEWTOT – CURTOT

6. Begin the SUBMIT.Click event routine by using Begincheck/Endcheck and Filecheck to ensure the department code exists. Your code should look like the following:

Begincheck Filecheck Field(#DEPTMENT) Using_File(deptab) Msgtxt('Department not fo Endcheck

Recall that the Endcheck error handling will branch to the Endroutine if the Filecheck raises an error.

7. Clear the working list EMPLOYS

```
Clr_List Name(#EMPLOYS)
```

8. Add an Exchange command to pass DEPTMENT and IIIPERCNT to the called function

Exchange Fields(#deptment #iiiprccnt)

- This will add these fields to the Exchange List which is passed to the called function.
- The called function will map these values into its variables and clear the

exchange list

9. The SRVCON field contains the value of the system variable *sserver_connected. The system variable is Y when connected to a server, and N when not connected to a server.

Add an If loop to call the function IIIFN21 locally when not connected.

```
If (#srvcon = N)
Call Process(*DIRECT) Function(iiifn21) Pass_Lst(#employs)
Endif
```

**Note:** The CALL command is passing the working list EMPLOYS to the called function, which will return it.

10. Add an Else to the If loop, which calls the function IIIFN21 on the server.

Your code should now look like the following:

```
If (#srvcon = N)
Call Process(*DIRECT) Function(iiifn21) Pass_Lst(#employs)
Else
Use Builtin(call_server_function) With_Args(*sserver_ssn 'iiifn21' Y *Default
If (#sstatus *NE OK)
Message Msgtxt('Call server function failed')
Endif
Endif
```

The CALL_SERVER_FUNCTION uses the following parameters:

#### WITH_ARGS

Server Symbolic Name	*sserver_ssn				
Name of RDML function to be called	IIIFN21				
Pass Current Exchange List	Y				
Receive Exchange List back	*default				
Working List 1	#EMPLOYS				
TO_GET					
Return Code	#SSTATUS				

- *sserver_ssn is a system variable which returns the server symbolic name of the server. The server symbolic name is defined by the CONNECT_SERVER BIF or by the automatic server connection to be used in this example.
- The called function IIIFN21 is called directly
- The (field) Exchange List is passed to IIIFN21. The Exchange List was set up by the Exchange command
- A returned Exchange List is not required in this case (Receive Exchange List back = *default)
- Only the first working list is passed to function IIFN21 and will be returned by IIIFN21.
- The Return Code will be work field SSTATUS
- 11. Complete the SUBMIT.Click event routine, with:
  - Clear the list view
  - Initialize fields CURTOT and NEWTOT
  - Read (SELECTLIST) the returned working list EMPLOYS
    - Accumulate CURTOT
    - Accumulate NEWTOT
    - Add and entry to list view
  - End read
  - Set IIITOTSAL to (NEWTOT CURTOT)

Your additional code should look like the following:

```
Endif
Clr_List Named(#LTVW_1)
#curtot #newtot := *zeroes
Selectlist Named(#employs)
#curtot += #salary
#newtot += #iiinewsal
Add_Entry To_List(#LTVW_1)
Endselect
```

#IIITOTSAL := (#newtot - #curtot) Endroutine

12. Compile your new form.

If required, see Appendix B. FRM095 for a complete solution.

## **Step 3. Test Salary Review Application**

- 1. Select form iiiCOM17 on the Last Opened tab. Use the right mouse menu to Execute it. Run it as a Windows Application. You are running the form locally on the desktop, with no connection to the IBM i server.
- 2. Enter a department code such as ADM. Enter a percentage figure, such as 5.50 and click the Submit button.
- 3. After a slight delay, the List View should be filled and the Cost of Increase field should contain a value. Your form has called function iiiFN21 locally.
- 4. Close the form.
- 5. Execute the form again, using the right mouse menu. Execute the form as a *Client to a RDMLX System i Server*.

Your form attachs to a job on the server, via the LANSA Listener.

6. Enter a department and percentage value and click the Submit button.

After a short delay the list view and Cost of Increase field should be populated.

Your form has called function iiiFN21on the server, using the CALL_SERVER_FUNCTION BIF. Function iiiFN21 has returned a working list of employees to the calling form.

7. Close your form.

### Summary

#### **Important Observations**

- Functions are called locally using the CALL command. The CALL can pass a working list into the called function.
- The called function must have a RCV_LIST() defined on its FUNCTION statement
- Passed working lists must be identically defined, in caller and called function.
- The EXCHANGE command enables an "exchange list" of fields to be passed into the called function.
- A called function uses EXCHANGE to return fields to the caller.
- Remote functions running on the server are called using CALL_SERVER_FUNCTION
- The CALL_SERVER_FUNCTION can handle passing and returning working lists and the fields exchange list.

## **Tips & Techniques**

- In a real application, the client/server connection would normally be established by a "connect" form at the start of the application.
- In this example we depended on the Visual LANSA IDE to start the connect to server.

## What You Should Know

- How to CALL a function
- How to use the EXCHANGE command to pass field values into a called function
- How to use CALL_SERVER_FUNCTION

# Appendix A. FRM095

#### Source code for function iiiFN21 – Calculate Salary Increase

Function Options(*DIRECT) Rcv_List(#employs) Def_List Name(#employs) Fields(#empno #fullname #deptment #salary #iiine⁻ Group_By Name(#empdata) Fields(#empno #surname #givename #deptment # Clr_List Named(#employs) Select Fields(#empdata) From_File(pslmst1) With_Key(#deptment) Nbr_Keys #iiinewsal := (#salary + (#salary * (#iiipercnt / 100))) #fullname := #surname + ', ' + #givename Add_Entry To_List(#employs) Endselect Return

# Appendix B. FRM095

## Source Code for Form iiiCOM17 – Salary Review

**Note:** component definitions (Define_Com . . .) have been omitted from this source to save space.

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #PRIM_FORM) Clientheight(463) Clientwidth Define_Com Class(#PRIM_GPBX) Name(#GPBX_1) Caption('Request') Disp . . . . Def_List Name(#employs) Fields(#empno #fullname #DEPTMENT #salary #i Define Field(#newtot) Reffld(#iiitotsal) Define Field(#srvcon) Type(*char) Length(1) Default(*sserver_connected) Define Field(#sstatus) Reffld(#io\$sts) Define Field(#curtot) Reffld(#iiitotsal) Evtroutine Handling(#com_owner.Initialize) Set Com(#com_owner) Caption(*component_desc) Endroutine Evtroutine Handling(#SUBMIT.Click) Begincheck Filecheck Field(#DEPTMENT) Using_File(deptab) Msgdta('Department not fc Endcheck Clr_List Named(#employs) Exchange Fields(#DEPTMENT #IIIPERCNT) If (#srvcon = N)Call Process(*DIRECT) Function(iiifn21) Pass Lst(#employs) Else Use Builtin(call server function) With Args(*sserver ssn 'iiifn21' Y *Default If (#sstatus *NE OK) Message Msgtxt('Call server function failed') Endif Endif Clr List Named(#LTVW 1) #curtot #newtot := *zeroes Selectlist Named(#employs) #curtot += #salary #newtot += #iiinewsal Add Entry To List(#LTVW 1)

Endselect #IIITOTSAL := (#newtot - #curtot) Endroutine End_Com

# FRM105 - Define a Trigger Function

#### **Exercise Concept**

- There is a business requirement that any salary increase which exceeds 1,000 must be approved by the HR Manager.
- A salary change could be made via more than one application.
- The solution is a file level trigger function, which runs before update, if the new salary exceeds current salary by more than 1,000.
- The trigger function, resets salary to its previous value, and sends a change request email to the HR Manager.
- The HR Manager makes a salary increase of more than 1,000 by changing Business Phone Number to A100 while changing salary. We are treating Business Phone as a 'spare field' for this exercise.
- The trigger function allows an increase of 1,000+ if Business Phone is A100. It also resets Business Phone to N/A in this case.
- In a real application the email message could contain the URL of a web application (a WAM) which the HR Manager simply clicks on to run and make the authorized change. To simplify this exercise you will make the change via a simple form.

# **Objectives:**

- To define a trigger for field salary on the file PSLMST
- The trigger event will be Before Update
- The trigger condition will be Salary greater than previous Salary
- The trigger function will send an email requesting change approval, if the increase in salary is greater than 1,000 and the business phone number is not equal to A100
- On receiving the email message, the salary increase will be made by the 'HR Manager' using a form. In a real application a WAM could be provided which may be executed using a URL in the email message, to confirm the salary change. A salary change greater than 1,000 is made by making Business phone number equal to A100 to indicate that the salary change is confirmed.

To achieve these objectives you must complete the following:

Step 1. Create a Trigger Function

Step 2. Define a Salary Trigger for Employee File (PSLMST)Step 3. Create an Employee Salary Change FormStep 4. Test your Employee Salary TriggerSummary

# **Step 1. Create a Trigger Function**

In this step you will create a file level trigger function for file PSLMST using a template. You will complete the trigger function based on the code supplied in Appendix A. FRM105. Note that you must change the email recipient to an email address to which you have access.

- 1. Create a Process iiiPRO02 iii Trigger Functions
- 2. Create a new function iiiFN03 Employee Salary Trigger, using template BBFILTRIG for file PSLMST
- 3. Remove all CASE, WHEN clauses except BEFUPD. Your code should look like the following:

Function Options(*DIRECT *NOMESSAGES *LIGHTUSAGE *MLOPTIMI Def_List Name(#TRIG_LIST) Type(*WORKING) Entrys(2) * Assume a "good" return initially Change Field(#TRIG_RETC) To('OK') Case Of_Field(#TRIG_OPER) * Handle a before update event When Value_Is('= BEFUPD') * Handle an event not catered for Otherwise Abort Msgtxt('File PSLMST trigger function invalidly invoked/ used.') Endcase * Return control to the invoker Return

Note: Comment lines have been removed.

- 4. Complete most of the trigger logic by copying the code provided in Appendix A. FRM105. This should be pasted after the Return command.
- 5. Review the supplied code.
  - Subroutine SEND_EMAIL generates and send and email message, using the LANSA Email BIFs
  - Subroutine ADD_TEXT is executed from SEND_EMAIL to create the email body text
  - In a real application, the URL inserted into the text body would be for web application (a WAM), which would will enable the salary change to be

confirmed simply by clicking on the URL to run it in the browser. To simplify this exercise, you will create a form to confirm the salary change.

- 6. Change the email recipient address to be any email address to which you have access.
- 7. Your email client may also require that the email originator address is a recognized address. Change this if necessary.
- 8. Complete the BEFUPD logic:
  - If the salary increase is greater than 1,000 and business phone is not equal A100. This value indicates the update needs to be approved.
    - Send an email message
    - Reset salary to previous value
  - else
    - Set business phone to N/A

#### Notes:

- Trigger functions receive a trigger list (working list TRIG_LIST) containing 0, 1 or 2 entries. TRIG_LIST contains the file data structure. That is, all file fields.
- For a file update, TRIG_LIST entry 1 contains the new record while entry 2 contains the old record.
- The TRIG_LIST is returned to the file OAM. This enables the trigger to modify field values, in a before-update trigger. This feature can be used to calculate virtual fields which cannot be produced using the standard virtual field derivation logic.

Your completed code should look like the following:

Case Of_Field(#TRIG_OPER) * Handle an before update event When Value_Is('= BEFUPD') Define Field(#increase) Reffld(#salary) Define Field(#newsal) Reffld(#salary) Get_Entry Number(1) From_List(#TRIG_LIST) #phonebusw := #phonebus #newsal := #salary Get_Entry Number(2) From_List(#TRIG_LIST) #increase := #newsal - #salary

```
If ((#increase *GT 1000) *And (#phonebusw *NE 'A100'))
Execute Subroutine(send_EMAIL)
* reset salary to previous value
Get_Entry Number(1) From_List(#TRIG_LIST)
#salary := #newsal - #increase
Upd_Entry In_List(#trig_list)
Endif
* reset business phone if contains A100
Get_Entry Number(1) From_List(#TRIG_LIST)
If (#phonebus = 'A100')
#phonebus := N/A
Upd_Entry In_List(#trig_list)
Endif
* Handle an event not catered for
Otherwise
Abort Msgtxt('File PSLMST trigger function invalidly invoked/ used.')
Endcase
* Return control to the invoker
Return
```

9. Compile your trigger function.

#### Step 2. Define a Salary Trigger for Employee File (PSLMST)

In this step you will define a Before Update trigger for field Salary, with a condition: salary greater than previous salary.

Note: You are defining a file level trigger which applies whenever this file is maintained. The field against which the condition is defined is not relevant, except that since it applies to **Salary** it makes sense to define the trigger on this field.

- 1. Open the file PSLMST in the editor.
- 2. Select the Rules and Triggers tab and add a new trigger, *Description*=**Salary** Change Trigger, for *Event*=**Before Update**
- 3. Set Trigger Function to **iiiFN03**

Details			
Description	Salary Ch	ange Trigger	
Function	iiifn03		
Sequence	1		\$
Trigger points			
	Before	After	
Open			
Close			
Read			
Insert			
Update	1		
Delete			

4. Add a trigger condition "Salary greater than previous Salary"

Compare to value SALARY	-
Compare to Value DAEART	-
Sequence 1	-

- 5. Save your changes.
- 6. Recompile the file, creating the OAM only

#### Step 3. Create an Employee Salary Change Form

In this step you will create a simple employee maintenance form for file PSLMST based on a template.

As noted earlier, an ideal solution for this application would be a 'Salary Change' WAM. The email message could be generated including the employee number (EMPNO) in the URL to be passed into the WAM when it runs from a link in the email message.

- 1. Create a new form iiiCOM18 Employee Maintenance. The form should be RDMLX enabled.
- 2. In the Source editor, delete the default code and run a template from the Template button on the Design ribbon. Select the template VL_BASEMNT and complete the template based on the following:

Supply a word which that describes WHAT this data entry program works with	Employee
Enter the name of the PHYSICAL file to be used by this template	PSLMST
How do you want to display the fields? Select from the types listed below	FF
Select fields	EMPNO, SURNAME, GIVENAME, PHONEBUS, SALARY

Adjust the size of the form. Your form should look like the following:

Employee		- • ×
Employee Number	ABCDE	
Employee Surname	ABCDEFGHIJKLMNOPQRST	
Employee Given Name(s)	ABCDEFGHIJKLMNOPQRST	
Business Phone Number	ABCDEFGHIJKLMNO	
Employee Salary	123,456,789.12	
<u> </u>	nsert Update	<u>D</u> elete
aAbBcCdDeEfFgGhHiljJkKlLmM	nNoOpPqQrRsStTuUvVwWxXyYzZ	▼▲

3. Compile your form.

# Step 4. Test your Employee Salary Trigger

To fully test your trigger, you require an email client such as Microsoft Outlook.

- In this step you will change an employee salary by more than 1,000. Your email client will warn you that a 'third party' is sending an email.
- Access your email account to which the message was sent. The Inbox should contain a message generated by the trigger function.
- Access the employee record and note that the trigger has prevented the change being made. Salary has not been changed.
- Make the required salary change by changing the business phone number to A100 at the same time as changing the salary. This time no email message will be generated
- Access the employee record again to confirm the salary has changed. Note that the trigger has set business phone number to N/A.
- 1. Ensure that your email client is started.
- 2. Execute form iiiCOM18. Fetch an employee such as A0090. Change salary by more than 1,000. The trigger function will be called, which will try to send an email. Your email client will display a warning dialog, which you should allow to continue.



3. Retrieve the email, which should look like the following:

a fer and a second and a second a s
HI,
PLOCCE PERD TOWN AT ANHAL PROFILED A COLOR OF ONE A OD
BLOGGS, FRED JOHN ALANhas received a salary increase of 2954.09
This exceeds allowable limits. Please confirm by entering
the new salary and set business phone to A100.
Click on the following link to confirm salary change
<pre> http://10.4.10.238/CGI-BIN/lansaweb?srve=JMIEMPSAL+ml=LANSA:XHTML+part=DEM+lang=ENG+f(EMPNO)=A0090 } </pre>
Purchased phase must be not to \$100 to confirm this change
Business Phone must be set to A100 to confirm this change.
Magazga gant automatically by Employee Trigger
Message sent automatically by Employee Trigger
المستحلين والمعين والمعاد والمتحاط المرودين والمحالية والمعاد والمحالية والمحالية والمحالين والمحالي والمحالية والم

The URL include illustrates how a WAM could be called to make the required salary change, passing in the required employee number (EMPNO).

4. Run form iiiCOM18 and make the required salary change, changing business phone number to A100 at the same time.

Note that this time an email message is not sent.

5. Retrieve this employee again, using form iiiCOM18. Note that the business phone number has been changed by the trigger function to N/A. This allows a further salary change to be correctly handled by the trigger function.

#### Summary

#### **Important Observations**

- File triggers are a powerful technique for implementing common application logic.
- Like all the other Repository techniques, using triggers will significantly simplify future system maintenance.
- All LANSA application programs maintaining the file will run the trigger if required.
- From V12, LANSA's Database Triggers enable non-LANSA programs to implement validation rules and triggers on LANSA-defined files

# Tips & Techniques

- Triggers are most efficient if they have an associated condition, meaning they are called only if required.
- A triggers function returns the trigger list to the OAM. This means that 'After Read' triggers can be used to calculate an 'undefined' virtual field.

# What You Should Know

- As well as running common logic, such as this "send Email" example, triggers can calculate virtual fields or modify real fields, as used in this trigger example. Of course these require the trigger to be defined on the correct event, such as Before Update.
- How to define a file level trigger and create and implement a file trigger function.
- How to use the CASE / WHEN / ENDCASE and USE commands.

# Appendix A. FRM105

#### Code to complete trigger function iiiFN02

Subroutine Name(SEND EMAIL) * COMMENT() Use Builtin(MAIL_START) * COMMENT(Set Mail Orginator Address) Change Field(#STD TEXT) To('training@lansa.co.uk') Use Builtin(MAIL ADD ORIGINATOR) With Args('Salary Trigger' #std tex £sts) Execute Subroutine(checksts) * COMMENT(Set mail recipient address) #std_textl := 'SMTP:anyone@acme.com' Use Builtin(MAIL_ADD_RECIPIENT) With_Args(TO 'JM Ivory' #std_textl) ' £sts) Execute Subroutine(checksts) * COMMENT() #std_descl := (#surname + '(' + #EMPNO + ') Exceeded allowed salary change' Use Builtin(MAIL_SET_SUBJECT) With_Args(#Std_descl) To_Get(#iof_sts) Execute Subroutine(checksts) * COMMENT() Execute Subroutine(ADD TEXT) * COMMENT() Use Builtin(MAIL SEND) To Get(#io£sts) Execute Subroutine(checksts) Endroutine Subroutine Name(checksts) If (#io£sts *NE OK) Message Msgtxt('Email service response was :' + #io£sts) Endif Endroutine * COMMENT() Subroutine Name(ADD_TEXT) * COMMENT() * COMMENT() Use Builtin(MAIL_ADD_TEXT) With_Args('HI,') Use Builtin(MAIL_ADD_TEXT) With Args(*BLANK) #fullname := #Surname + ',' + #givename

#std instr := (#Fullname + 'has received a salary increase of ' + #increase.asstri Use Builtin(MAIL_ADD_TEXT) With_Args(#std_instr) #std instr := 'This exceeds allowable limits. Please confirm' Use Builtin(MAIL_ADD_TEXT) With_Args(*BLANK) Use Builtin(MAIL ADD TEXT) With Args(*BLANK) #std instr := 'Click on the following link to confirm salary change' Use Builtin(MAIL_ADD_TEXT) With_Args(#std_instr) Use Builtin(MAIL ADD TEXT) With Args(*BLANK) Use Builtin(MAIL ADD TEXT) With Args(*BLANK) * COMMENT(URL for WAM to confirm salary change) * COMMENT(Change service name using your initials) #std_gsel := ('http://localhost/CGI-BIN/lansaweb? srve=iiiEMPSAL+ml=LANSA:XHTML+part=DEM+lang=ENG+f(EMPNO)= Use Builtin(MAIL_ADD_TEXT) With_Args(#std_qsel) Use Builtin(MAIL_ADD_TEXT) With_Args(*BLANK) #std instr := 'Set Business Phone to A100 to confirm' Use Builtin(MAIL_ADD_TEXT) With_Args(#std_instr) Use Builtin(MAIL ADD TEXT) With Args(*BLANK) #std_instr := 'Message sent automatically by Employee Trigger' Use Builtin(MAIL ADD_TEXT) With_Args(#std_instr) Endroutine

## FRM115 - Writing Reports

#### **Objective:**

- To show how Forms and Functions can interoperate.
- To use the CALL and EXCHANGE commands.
- To introduce the SUBMIT command.
- To explain how to enable functions for Full RDMLX.
- To introduce the LANSA reporting templates.
- To highlight the commands used to create a reporting function in LANSA.
- To create a very simple report listing the sections in a department by executing a template.
- To manually create an employee report listing the salary details for each department.
- To introduce the following reporting commands:
  - DEF_LINE
  - DEF_HEAD
  - DEF_FOOT
  - DEF_BREAK
  - DEF_REPORT
  - PRINT
  - SKIP
  - SPACE
  - ENDPRINT
  - KEEP_XXXX (KEEP_AVG, KEEP_COUNT, KEEP_MAX, KEEP_MIN, KEEP_TOTAL).

To achieve these objectives you must complete the following steps:

- Step 1. Create a Simple List Style Report Using a Template
- Step 2. Call a Function
- Step 3. Enable For Full RDMLX
- Step 4. Manually Create a Reporting Function
- Step 5. Add a Header and Footer

Step 6. Keep Statistics and PrintStep 7. Add a Grand Total LineSummary

### **Before You Begin**

You may wish to refer to the following topics:

- In the *Visual LANSA Developer Guide*: Creating Applications using Functions and Producing Reports using LANSA.
- In the *Technical Reference Guide* RDMLX Commands and RDMLX Features.

In order to complete this exercise, you should have completed the previous exercises.

#### Step 1. Create a Simple List Style Report Using a Template

In this step, you will create a simple list style report on the Section file. A simple function will be created by executing a LANSA template. In later steps in this exercise, you will code a similar reporting function. Reporting commands are not supported from LANSA forms. You need to code reports using LANSA functions.

1. Create a process iiiPRO03 Reporting Process (where iii are your course assigned initials). If you are using iii=DEM, then your component must be named DEMPRO01. Do not open the process in the editor.

New Process	×
iiiPRO03	Create
Training Reports	Cancel
IIIPRO03	Cancer
	Open in Editor
	iiiPRO03 Training Reports

2. Create a function **iiiFN02 Section Report** belonging to your process iiiPRO03. Select the template **FRPRT01 List Style Report with Generic Selection**. The function should **NOT** be Enabled for RDMLX since it creates an entry panel, using a REQUEST command, which cannot be used in an RDMLX Enabled function.

Open the function in the editor in order to allow the template to execute.

	New Function		×
Process Name	iiiPRO03	 Create	1
Description	Training Reports	Cancel	1
Name	iiiFN02	concer	J
Description	Section Report	Open in Editor	
Template	FRPRT01 - List Style Report with Generic Selection		
Identifier	IIIFN02		
Enabled For RDMLX			

3. Answer the template questions as shown in the table below. Use the online help if you need additional information about the answer to each template question.

TEMPLATE QUESTION	ANSWER
Do you want this function to be part of an action-bar style process?	Ν
Enter the name of the primary file to be used by this template	SECTAB
Select related files	Select DEPTAB
Fields to appear in header line	Select DEPTMENT and DEPTDESC
Fields to appear in detail line	Select SECTION, SECDESC, SECADDRR1
Fields to trigger new page	Select DEPTMENT
Please specify a title for this report	Section Listing by Department
Do you want this report to run in batch	Y

4. The template generated code should appear something like the following:

FUNCTION OPTIONS(*DEFERWRITE *DIRECT) **OPEN USE_OPTION(*ONDEMAND)** GROUP_BY NAME(#FETCHDATA) FIELDS(#SECTION #SECDESC #SEC DEF_HEAD NAME(#HDR01) FIELDS(#REP1PAGE #DATE #TIME #FUN( #DEPTDESC) TRIGGER_BY(*OVERFLOW #DEPTMENT) DESIGN(*DO' DEF LINE NAME(#DET01) FIELDS(#SECTION #SECDESC #SECADDR1 CHANGE FIELD(#STD_TITLE) TO("Section Listing By Department") * If this program is running online IF COND('*JOBMODE = I') * Request report print criteria REQUEST FIELDS(#DEPTMENT #SECTION) DESIGN(*DOWN) IDENTI * Submit batch run of this program SUBMIT PROCESS(#PROCESS) FUNCTION(#FUNCTION) EXCHANGE( JOB(#FUNCTION) * Else, if this program is running in batch ELSE

```
* Select required SECTAB details
SELECT FIELDS(#FETCHDATA) FROM_FILE(SECTAB) WITH_KEY(#DI
NBR_KEYS(*COMPUTE) GENERIC(*YES)
* Fetch file DEPTAB details
FETCH FIELDS(#FETCHDATA) FROM_FILE(DEPTAB) WITH_KEY(#DE)
* Print the detail line
PRINT LINE(#DET01)
ENDSELECT
* Finish all printing and end program
ENDPRINT
ENDIF
```

- 5. Compile the process and function.
- 6. Execute the process that contains your reporting function.
  - a. When you execute your function, notice the Default Printer setting. The value *PATH will output the report as a file in the partition directory. For example C:\Program Files\LANSA\X_WIN95\X_LANSA\X_PPP where PPP is your partition. The file will have the same name as the function with a sequentially numbered file extension. e.g. demfn02.001. Refer to the Technical Reference Guide for more information on this topic.

Process as Win	ndows Ap – 🗆 🛛 🗡
Default Printer	*PATH
Debug	N
Debug Host	johni_win8:51237
Trace	N
Max Trace Lines	20000
Trace level	4
Trace Categories	ALL
Heap Validation	X
Show Command	Line N
OK Cancel	Help Parameter Help

b. When the process menu appears, select the **Section Report** by doubleclicking. Your function should appear something like the following:

Section Report	
Department Code	
OK Prompt Msgs Exit Cancel	Help

- c. Enter a department of ADM to list all Sections in the Administration department. (If you do not enter a department or section, a list of all records in the Section file will be created.)
- d. Run the report by clicking on the OK button.
- e. Open your report file in Notepad to view it. It should look like the following:

- f. Execute your reporting function again. Enter a department of ADM and a section of 01.
- g. Execute your reporting function again. Enter a department of A. You should see listings for both the ADM and AUD departments.

# Step 2. Call a Function

In this step, you will create a form that will be used to call the reporting function. The purpose of this step is to show how forms and functions can work together. In your first version, the form will simply call the function directly. (You can also invoke the process menu.)

- 1. Create a form named iiiCOM19 Submit Report (where iii are your course assigned initials). If you are using iii=DEM, then your component must be named DEMCOM19.
- 2. Drag and drop a push button onto the form.
  - a. Set the button *Name* to REPORT.
  - b. Set the button *Caption* to Generate Report.
  - c. Create a *Click* event routine for the REPORT button.
- 3. Drag and drop the DEPTMENT and SECTION fields onto the form.
- 4. Drag and drop a status bar onto the form.
- 5. Your finished form might appear something like the following.

																x	
Department Code	ABCD			· · ·		· · · · · · · · · · · · · · · · · · ·	· · ·		· · ·			· · ·	· · ·	 	•••••		
Section Code	AB		· · · · ·	· · · · ·			G	ien	erat	te R	lep	ort			• • • • •		· · · · ·
	· · · · · · · · · · · · · · · · · · ·			· · · · ·		· · · · ·			· · ·			· · ·	· · ·	 	• • • • •		
aAbBcCdDeEfFgGhHiIjJkKILm	MnNoOpP	qQr	RsS	tTu	Uv	Vw	Wx	Xy١	/zZ					•	ŀ	•	

6. In the *REPORT.Click* event routine, simply call the iiiFN02 reporting function. Review the parameters for the CALL command.

Your finished code should appear as follows:

```
EVTROUTINE HANDLING(#REPORT.Click)
CALL PROCESS(*DIRECT) FUNCTION(iiiFN02)
ENDROUTINE
```

- 7. Compile and execute the form.
  - a. Press the Generate Report button on the form.

- b. The iiiFN02 Section Report will be executed. Notice that you cannot set focus back to the calling form. Press the EXIT button. Notice that the Form is also closed when the Function is exited. The EXIT_USED parameter on the CALL command can be used to control how the form responds when the function ends.
- c. Execute the form again. Enter a Department Code of ADM and press the Generate Report button on the form. Notice that the DEPTMENT value is not passed to the function. (The function fields are both blank.)
- d. Enter a Department Code of ADM and press OK to submit the report. Notice that the form is not closed once the report is submitted. Notice that a message appears indicating that the report was submitted.
- e. Close the form.
- 8. In the *REPORT.Click* event routine, use the EXCHANGE command to pass the values of the DEPTMENT and SECTION fields to the function.

Your finished code should appear as follows:

EVTROUTINE HANDLING(#REPORT.Click) EXCHANGE FIELDS(#DEPTMENT #SECTION) CALL PROCESS(*DIRECT) FUNCTION(iiiFN02) ENDROUTINE

- 9. Compile and execute the form.
  - a. Enter a Department Code of ADM and a Section Code of 01. Press the Generate Report button.
  - b. The iiiFN02 Section Report will be executed. Notice that the function now shows the values that were passed from the form.
  - c. Exit the function without submitting the report.
  - d. By using the EXCHANGE command, the DEPTMENT and SECTION fields are passed to the reporting function. The reporting function no longer requires a screen to request the DEPTMENT and SECTION fields. In the next step, you will modify the iiiFN02 function and remove the REQUEST.
- 10. In the *REPORT.Click* event routine, use the SUBMIT command instead of the CALL to invoke the report function and pass the values of the DEPTMENT and SECTION fields.

Your finished code should appear as follows:

#### EVTROUTINE HANDLING(#REPORT.Click) SUBMIT PROCESS(iiiPRO01) FUNCTION(iiiFN02) EXCHANGE(#DEPTM ENDROUTINE

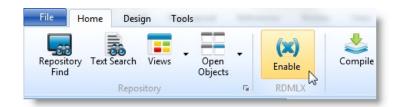
- 11. Compile and execute the form.
  - a. Enter a Department Code of ADM. Press the Generate Report button.
  - b. Notice that the REQUEST is not displayed and the report function is submitted to batch for execution. Also notice that the form can be used immediately after the function is submitted. The form does not have to wait for the function to return control because the function has been submitted and is executing in batch.

# **Step 3. Enable For Full RDMLX**

In this step, you will modify the iiiFN02 Section Report so that it does not request the DEPTMENT and SECTION fields. You will also enable the iiiFN02 Reporting function for Full RDMLX so that you can use the full set of RDMLX objects and features.

- 1. Make sure that the iiiFN02 Section Report function is opened in the editor.
- 2. Choose the RDMLX command in the Home tab of the ribbon to set the function as RDMLX enabled.

**Note:** You cannot undo this change. Once set as Full RDMLX, the function cannot be changed back.



3. Perform a function check of the code. An error will appear because the REQUEST command cannot be used in an RDMLX enabled Function.

PRC0010/FFC	0 • E • 864W DISPLAY, REQUEST and POP_UP Completed : +0 warning messages iss
<ul> <li>▲ Error</li> <li>● PRC0047/LII0</li> <li>● PRC0010/FFC</li> <li>● PRC0011/FFC</li> </ul>	
<ul> <li>▶ PRC0047/LII0</li> <li>▶ PRC0010/FFC</li> <li>▶ PRC0011/FFC</li> </ul>	
	Completed : +1 fatal messages issued
Definitions	
Function Design	16
I/O Operations	
🖻 🛃 Report Operatio	ons

4. Delete the unneeded code in the iiFN02 function.

You can delete all of the highlighted lines:

FUNCTION OPTIONS(*DEFERWRITE *DIRECT) OPEN USE_OPTION(*ONDEMAND)

GROUP BY NAME(#FETCHDATA) FIELDS(#SECTION #SECDESC #SEC DEF_HEAD NAME(#HDR01) FIELDS(#REP1PAGE #DATE #TIME #FUN( #DEPTDESC) TRIGGER BY(*OVERFLOW #DEPTMENT) DESIGN(*DO' DEF LINE NAME(#DET01) FIELDS(#SECTION #SECDESC #SECADDR1 CHANGE FIELD(#STD_TITLE) TO("Section Listing By Department") * If this program is running online IF COND('*JOBMODE = I') * Request report print criteria REQUEST FIELDS(#DEPTMENT #SECTION) DESIGN(*DOWN) IDENTI * Submit batch run of this program SUBMIT PROCESS(#PROCESS) FUNCTION(#FUNCTION) EXCHANGE( JOB(#FUNCTION) * Else, if this program is running in batch ELSE * Select required SECTAB details SELECT FIELDS(#FETCHDATA) FROM_FILE(SECTAB) WITH_KEY(#DI NBR KEYS(*COMPUTE) GENERIC(*YES) * Fetch file DEPTAB details

FETCH FIELDS(#FETCHDATA) FROM_FILE(DEPTAB) WITH_KEY(#DE * Print the detail line

PRINT LINE(#DET01)

ENDSELECT

* Finish all printing and end program

ENDPRINT

ENDIF

You may now use assign statements and expressions in your code

FUNCTION OPTIONS(*DEFERWRITE *DIRECT) OPEN USE_OPTION(*ONDEMAND) GROUP_BY NAME(#FETCHDATA) FIELDS(#SECTION #SECDESC #SEC DEF_HEAD NAME(#HDR01) FIELDS(#REP1PAGE #DATE #TIME #FUN( #DEPTDESC) TRIGGER_BY(*OVERFLOW #DEPTMENT) DESIGN(*DO' DEF_LINE NAME(#DET01) FIELDS(#SECTION #SECDESC #SECADDR1 #STD_TITLE := 'Section Listing By Department' * Select required SECTAB details SELECT FIELDS(#FETCHDATA) FROM_FILE(SECTAB) WITH_KEY(#DI NBR_KEYS(*COMPUTE) GENERIC(*YES) * Fetch file DEPTAB details FETCH FIELDS(#FETCHDATA) FROM_FILE(DEPTAB) WITH_KEY(#DE * Print the detail line PRINT LINE(#DET01) ENDSELECT * Finish all printing and end program ENDPRINT

Notice that the STD_TITLE has been changed to use the Full RDMLX syntax and can now use an assignment command.

- 5. Compile the iiiFN02 function.
- 6. Close the function in the editor.
- 7. Execute your iiiCOM19 form and generate a test report.

# **Step 4. Manually Create a Reporting Function**

In this step, you will manually create the report in order to learn how to use the reporting commands.

The report will simply list all employees in a department and total the salary information.

The finished report will appear something like the following:

📺 eomfn05.002.txt - Notepad	
<u>F</u> ile <u>E</u> dit F <u>o</u> rmat <u>V</u> iew <u>H</u> elp	
Page 1 Function IIIFN05 Title Employee Report Department ADM Description ADMINISTRATOR DEPT	Į
Section Employ Start Code Number Salary Date Ol Al001 2,345.82 9/08/92	
01 A1012 26,456.04 1/05/86	- 1
A A A A A A A A A A A A A A A A A A A	E
Employee Count 20 Total Amount 2,916,027.38	
Date 9/11/10 Time 16:02:21 Page 100 Page 2 Function IIIFN05 Title Employee Report Department AUD Description INTERNAL AUDITING	2
Section Employ Start Code Number Salary Date Ol Al007 26,780.04 1/03/85	
01 A1008 450,000.04 1/12/86	

A new page will be printed for each Department. A count of the employee and the total of the SALARY field will be printed for each department. A grand total of all departments will be printed on the last page of the report. The report will be 80 characters wide and 66 lines long.

Before you start, it is recommended that you review Producing Reports in the *Developer Guide*.

1. You will build the reporting function in small steps. In this step, you will start coding a reporting function:

Define the report to be 66 lines long and 80 characters wide

Define a report line (#DET01) containing SECTION, EMPNO, SALARY and STARTDTE

Select all records which generically match the Department Code and Section Code based on the number of keys entered

Print the report line (#DET01)

End the select loop

End the print.

- 2. Create a new function iiiFN04 Manual Report belonging to process iiiPRO01. Create the function without using a template. It should be an RDMLX enabled function.
- 3. Try to add the necessary code based on the function description above.
- 4. Your code should appear as follows:

FUNCTION OPTIONS(*DIRECT) DEF_REPORT FORMSIZE(66 80) GROUP_BY NAME(#REPDATA) FIELDS(#DEPTMENT #DEPTDESC #SE #STARTDTE) DEF_LINE NAME(#DET01) FIELDS(#SECTION #EMPNO #SALARY #ST SELECT FIELDS(#REPDATA) FROM_FILE(PSLMST1) WITH_KEY(#DEF NBR_KEYS(*COMPUTE) GENERIC(*YES) PRINT LINE(#DET01) ENDSELECT ENDPRINT

- 5. Save the code and then full function check or build the RDML. Make any corrections, if required, and then resave and check the function.
- 6. Compile the process and function. Be sure that the function is debug enabled.
- 7. Change your Submit Report form iiiDEM19 to submit function iiiFN04 and compile it.

- 8. Submit your new report function using form iiiCOM19.
  - a. Enter a department of ADM to list all Sections in the Administration department.
  - b. Submit the report function again. Enter a department of A and leave the section blank. Notice that there is no break in the report pages when the department code changes. You will correct this in the next step.

# Step 5. Add a Header and Footer

In this step, you modify the function to include a header and footer. You will need to make the following changes:

- Add a header to the report. Include the report page number (REP1PAGE), function name (FUNCTION), a title (STD_TITLE), DEPTMENT and the DEPTDESC fields. List the fields down the page. Specify that a new page should be printed whenever a page is full (each new page always has a header) and when the department code changes.
- Change the STD_TITLE field to be Employee Report.
- Add a footer to the report which lists the date (DATE), time (TIME), and report page number (REP1PAGE) across the bottom of the page.
- In the SELECT loop, FETCH the DEPTDESC field from DEPTAB using the department code.

Remember to include KEEP_LAST logic.

- 1. Review the repository field definitions for the REP1PAGE, DATE, TIME and FUNCTION fields.
- 2. Try to make the necessary code changes based on the function description above.
- 3. Your code should appear as follows:

FUNCTION OPTIONS(*DIRECT) DEF_REPORT FORMSIZE(66 80) GROUP_BY NAME(#REPDATA) FIELDS(#DEPTMENT #DEPTDESC #SE #STARTDTE) DEF_HEAD NAME(#HDR01) FIELDS(#REP1PAGE #FUNCTION #STD_T TRIGGER_BY(*OVERFLOW #DEPTMENT) DESIGN(*DOWN) DEF_FOOT NAME(#FTR01) FIELDS(#DATE #TIME #REP1PAGE) DEF_LINE NAME(#DET01) FIELDS(#SECTION #EMPNO #SALARY #ST #STD_TITLE := 'Employee Report' SELECT FIELDS(#REPDATA) FROM_FILE(PSLMST1) WITH_KEY(#DEF NBR_KEYS(*COMPUTE) GENERIC(*YES) FETCH FIELDS(#DEPTDESC) FROM_FILE(DEPTAB) WITH_KEY(#DEP' PRINT LINE(#DET01) ENDSELECT ENDPRINT

- 4. Save the code and then full function check or build the RDML Make any corrections as required. Exit the editor.
- 5. Compile the function. Be sure that the function is debug enabled.
- 6. Submit your new function using form iiiDEM19.
  - a. Enter a department of A and leave the section blank.
  - b. Check that the report headers and footers properly appear.

# **Step 6. Keep Statistics and Print**

In this step, you will keep some statistics based on each department and define some report breaks.

- Define a working field for the report called TOTAL. The field should be P(14,2) with an edit code of A. Enter a label, column heading and description of Total Amount.
- Define a working field for the report called EMPCOUNT. The field should be P(3,0) with an edit code of A. Enter a label, column heading and description of Employee Count.
- Define a report break which prints the EMPCOUNT and TOTAL as a trailing break when the DEPTMENT code changes.
- Keep track of the total of the salary amounts for each department in the TOTAL field.
- Keep a count of the number of employees in each department in the EMPCOUNT field.
- 1. Try to make the necessary code changes based on the function description above.
- 2. Your code should appear as follows:

FUNCTION OPTIONS(*DIRECT) DEFINE FIELD(#TOTAL) TYPE(*DEC) LENGTH(14) DECIMALS(2) LAB DESC('Total Amount') COLHDG('Total Amount') EDIT_CODE(A) DEFINE FIELD(#EMPCOUNT) TYPE(*DEC) LENGTH(3) DECIMALS(0) ] DESC('Employee Count') COLHDG('Employee Count') EDIT CODE(A) DEF REPORT FORMSIZE(66 80) GROUP BY NAME(#REPDATA) FIELDS(#DEPTMENT #DEPTDESC #SE **#STARTDTE**) DEF_HEAD NAME(#HDR01) FIELDS(#REP1PAGE #FUNCTION #STD_T TRIGGER BY (*OVERFLOW #DEPTMENT) DESIGN (*DOWN) DEF FOOT NAME(#FTR01) FIELDS(#DATE #TIME #REP1PAGE) DEF LINE NAME (#DET01) FIELDS (#SECTION #EMPNO #SALARY #ST. DEF BREAK NAME(#BRK01) FIELDS(#EMPCOUNT #TOTAL) TRIGGE] **#STD TITLE** := 'Employee Report' SELECT FIELDS(#REPDATA) FROM_FILE(PSLMST1) WITH_KEY(#DEF NBR KEYS(*COMPUTE) GENERIC(*YES) KEEP TOTAL OF FIELD(#SALARY) IN FIELD(#TOTAL) BY FIELD(#D

#### KEEP_COUNT OF_FIELD(#EMPNO) IN_FIELD(#EMPCOUNT) BY_FIEL FETCH FIELDS(#DEPTDESC) FROM_FILE(DEPTAB) WITH_KEY(#DEP PRINT LINE(#DET01) ENDSELECT ENDPRINT

- 3. Save the code and then full function check or build the RDML. Make any corrections as required.
- 4. Exit the editor.
- 5. Compile the function.
- 6. Submit your new function using form iiiCOM19.
  - a. Enter a department of A and leave the section blank.
  - b. Check that the break totals are properly printed and check that they are correct. Each department should have its own totals and employee count.

# Step 7. Add a Grand Total Line

- 1. In this step, you will add the grand total of all departments displayed in the report and you will add a blank line between printed lines:
  - a. Define a working field for the report called GRANDTOT. The field should be P(17,2) with an edit code of A. Enter a label, column heading and description of Grand Total.
  - b. Define another report break which prints the GRANDTOT field at the end of the report.
  - c. Add another KEEP_TOTAL command to accumulate SALARY into the GRANDTOT field.
  - d. Add a blank line (SPACE) between each printed line.
- 2. Try to make the necessary code changes based on the function description above.

Your code should appear as follows:

FUNCTION OPTIONS(*DIRECT)

DEFINE FIELD(#TOTAL) TYPE(*DEC) LENGTH(11) DECIMALS(2) LAB DESC('Total Amount') COLHDG('Total Amount') EDIT_CODE(A) DEFINE FIELD(#EMPCOUNT) TYPE(*DEC) LENGTH(3) DECIMALS(0) ] DESC('Employee Count') COLHDG('Employee Count') EDIT_CODE(A) DEFINE FIELD(#GRANDTOT) TYPE(*DEC) LENGTH(17) DECIMALS(2) DESC('Grand Total') COLHDG('Grand Total') EDIT CODE(A) DEF REPORT FORMSIZE(66 80) GROUP BY NAME(#REPDATA) FIELDS(#DEPTMENT #DEPTDESC #SE **#STARTDTE**) DEF HEAD NAME(#HDR01) FIELDS(#REP1PAGE #FUNCTION #STD T TRIGGER BY(*OVERFLOW #DEPTMENT) DESIGN(*DOWN) DEF FOOT NAME(#FTR01) FIELDS(#DATE #TIME #REP1PAGE) DEF_LINE NAME(#DET01) FIELDS(#SECTION #EMPNO #SALARY #ST. DEF_BREAK NAME(#BRK01) FIELDS(#EMPCOUNT #TOTAL) TRIGGE] DEF_BREAK NAME(#BRK02) FIELDS(#GRANDTOT) **#STD_TITLE := 'Employee Report'** SELECT FIELDS(#REPDATA) FROM_FILE(PSLMST1) WITH_KEY(#DEP NBR KEYS(*COMPUTE) GENERIC(*YES) KEEP TOTAL OF FIELD(#SALARY) IN FIELD(#GRANDTOT) KEEP TOTAL OF FIELD(#SALARY) IN FIELD(#TOTAL) BY FIELD(#D

KEEP_COUNT OF_FIELD(#EMPNO) IN_FIELD(#EMPCOUNT) BY_FIEL FETCH FIELDS(#DEPTDESC) FROM_FILE(DEPTAB) WITH_KEY(#DEP' PRINT LINE(#DET01) SPACE ENDSELECT ENDPRINT

- 3. Save the code and then full function check or build the RDML. Make any corrections as required.
- 4. Compile the function.
- 5. Submit your new function.
  - a. Enter a department of A and leave the section blank.
  - b. Check that the report grand total is correct and check that it appears in the proper location.
- 6. Your report should look like the following:

```
eomfn05.002.txt - Notepad
File Edit Format View Help
Page .....
                   1
Function ..... IIIFN05
Title..... Employee Report
Department.... ADM
Description.... ADMINISTRATOR DEPT
    Section
             Employ
                                   Start
             Number
                           Salary Date
    Code
    01
             A1001
                          2,345.82 9/08/92
             A1012
    01
                         26,456.04 1/05/86
    01
             A1013
                         78,977.04 1/12/85
                        313,000.04 12/12/84
    01
             A1015
             A1020
                        121,500.04 1/02/88
    01
             A1021
                          87,000-04 1/03/80
             A1014
                         68,000.04 1/02,
    02
             A2001
                         28,999.00 1/05/10
    02
             A1004
                         21,000.04 1/05/80
    03
    03
             A2002
                         26,888.00 1/04/10
                         30,000.00 31/05/95
             A3564
    04
                         35,000.04 3/07/89
    05
             A0193
                                   2,916,027.38
Employee Count 20
                     Total Amount
```

#### 7. OPTIONAL: Paint the Report

If you wish, you may paint the report using the Report Painter. The Report Painter is very similar to the LANSA Screen Painter and is accessed from the function editor.

Report	ository Help Cro	oss References							
🔅 Add 👻 💥	🔍 🔍 Curi	rent Report 1	👻 Cyc	les 2 🔻 R	epeat 2	-	Options	; •	
-Command Text									
Def_Line	Name(#DETC	01) Fields(#	SECTION	I #EMPNO #	SALARY	#STAP	RTDTE)	1	
	5	10 15	20	25 30	35	40	45	50	55
	Page		1234						
	Function	n	ABCDEF	G					
HDR01				dDeEfFgGI	HiljJ	kKlLmM	ínNoO _I	PqQrl	RsStT
		ent							
	Descrip	tion	ABCDEF	GHIJKLMN	PQRST				
	titutititititititi								
DET01	Sec	tion E	mploy			Start			
	Cod	de N	umber	Sa.	lary	Date			
	AB		CDE 1	23,456,71	39.12	12/34/	56		
DET01	h -								
DET01 DET01	AB	AB	CDE 1	23,456,7		12/34/			
	AB				89.12		/56	L2CR	
	AB			23,456,7; otal Amo	89.12		/56	L2CR	
DET01	AB Employee	e Count	123CR T	otal Amo	39.12 int 12		/56	L2CR	
DET01	AB Employee	e Count	123CR T		39.12 int 12		/56	L2CR	
DET01 BRK01	AB Employee	e Count	123CR T	otal Amo	39.12 int 12		/56	L2CR	
DET01 BRK01 BRK02	AB Employee	e Count	123CR T	otal Amo	39.12 int 12		/56	L2CR	
DET01 BRK01	AB Employee	e Count otal 123	123CR T	otal Amo	39.12 int 12 5.12CR		/56	L 2CR	
DET01 BRK01 BRK02 DET01	AB Employee Grand To Sect Coo	e Count otal 123 tion E: de N	123CR T ,456,78 mploy umber	otal Amo 9,012,34 Sa.	39.12 int 12 5.12CR lary	3,456, Start Date	~56 .789.1	L2CR	
DET01 BRK01 BRK02 DET01 DET01	AB Employee Grand To Sect Coo AB	e Count otal 123 tion E de N AB	123CR T ,456,78 mploy umber CDE 1	otal Amo 9,012,34 Sa. 23,456,71	39.12 int 12 5.12CR Lary 39.12	3,456, Start Date 12/34/	<pre>/56 .789.1 .789.5 .756</pre>	L2CR	
DET01 BRK01 BRK02 DET01	AB Employee Grand To Sect Coo	e Count otal 123 tion E de N AB	123CR T ,456,78 mploy umber CDE 1	otal Amo 9,012,34 Sa.	39.12 int 12 5.12CR Lary 39.12	3,456, Start Date 12/34/	<pre>/56 .789.1 .789.5 .756</pre>	L2CR	

The report painter displays all screen elements, and enables one of these to be edited at one time. For example report header (HDR01) or report detail line (DET01). Changes made in the painter will be reflected in the RDML source.

## Summary

## **Important Observations**

- The CALL and SUBMIT commands will execute a function. When a function is called, the calling program is not accessible until after the called program ends.
- EXCHANGE can only be used to pass information to a function. It can be passed from a form or from a function to another function.
- Once a function has been enabled for Full RDMLX, it cannot be changed back to an RDML Function.
- Reporting commands can only be used in LANSA Functions. They cannot be used in Forms.
- LANSA provides a very simple set of reporting commands that can be combined to create very powerful reports.

# **Tips & Techniques**

• In order for the report to print, LPT1 needs to be mapped to a particular printer for all NT-based systems. Therefore the following command needs to be issued from the command line:

NET USE LPT1: <PATH>\<printer name>

For example:

NET USE LPT1: \\NT1\HP

# What You Should Know

- How to CALL a function from a form.
- How to SUBMIT a function to batch.
- How to EXCHANGE information between LANSA programs.
- How to enable a function for RDMLX.
- What some of the differences are between RDML and RDMLX functions.
- You should be familiar with the following reporting commands:
  - DEF_LINE
  - DEF_HEAD
  - DEF_FOOT
  - DEF_BREAK

- DEF_REPORT
- PRINT
- SKIP
- SPACE
- ENDPRINT
- KEEP_XXXX (KEEP_AVG, KEEP_COUNT, KEEP_MAX, KEEP_MIN, KEEP_TOTAL)

# FRM125 - Check Out / In to IBM i

## **Objectives:**

- To introduce the concepts of Master and Slave development in a distributed development environment.
- To learn how to refresh object lists from the LANSA for iSeries Master Repository.
- To learn how to check out objects from the LANSA for iSeries Master Repository.
- To learn how to check in objects to the LANSA for iSeries Master Repository.
- To learn how to delete objects locally and in the LANSA for iSeries Master Repository.

In order to complete this exercise, you must meet the following requirements:

- You must have task tracking properly configured on LANSA for iSeries in order to work with the Visual LANSA environment.
- You must have a valid user profile to access the IBM i and the LANSA for iSeries development environment.
- You must have a valid task ID to use for development on LANSA for iSeries.
- You must have a properly installed and configured Visual Slave System with a working connection to the LANSA for iSeries Master System.
- You must be a licensed Visual LANSA developer and you must have the proper LANSA for iSeries licenses installed that allow both IBM i and Visual LANSA development. You cannot complete this exercise if you are using a trial (unlicensed) version of Visual LANSA.

Note: This exercise cannot be completed if you are using an Independent Visual LANSA System.

To achieve these objectives you must complete the following:

Step 1. Create a LANSA for iSeries Object

Step 2. Refresh Objects in Visual LANSA

Step 3. Check Out Object

Step 4. Check In Changes

Step 5. Delete from Repository

#### Summary

# **Before You Begin**

In order to complete this exercise, it is recommended that you complete the preceding exercises or are at least familiar with the Visual LANSA user interface.

You may wish to review the following topics:

In the *User Guide*, review the Check In Tab and Check Out Tab and the Propagation Tab.

In the *Administrator Guide*, review Host Monitor.

# Step 1. Create a LANSA for iSeries Object

In this step, you will logon to LANSA for iSeries using any partition, where you will create a new system variable that you will amend using Visual LANSA. A System Variable may be created using LANSA for iSeries, in either an RDML or RDMLX enabled partition

- 1. Logon to LANSA for iSeries using your IBM i developer profile and task ID.
- 2. From the *LANSA Main System Menu*, select the option to *Work with System Variables*. The list displayed is **NOT** partition dependent. It is a system-level list; that is, it will be the same in each partition.
- 3. Press *F6* to create a new system variable as follows:

System Variable Name:*AUTOALP09iiiNUM where iii=your initialsDescription:Next Available NumberMethod of Derivation:DYNAMICData Type:ALPHALength:7Program Name:M@SYSNUMProgram Type:GL

Use the online help text search facilities (extended help) and review *System Variables*. Scroll to the end of the help to review the information on the ***AUTOALPnnxxxxxxx** system variable. This is a special 'data area' system variable layout which is used in conjunction with a LANSA shipped evaluation program **M@SYSNUM**. The system variable retrieves a number **nn** (in this case it should be 9 but you have entered 7 - it will be changed to 9 using Visual LANSA) long from data area **xxxxxxxxx**, increments it, updates the data area and returns it as an alphanumeric value.

- 4. Press *Enter* to save the system variable definition.
- 5. Press *F12* to exit the Add System Variable Definition panel.

Note: The data area and the value of the system variable will automatically

be defined when the system variable is first accessed.

6. Exit the LANSA for iSeries System.

# Step 2. Refresh Objects in Visual LANSA

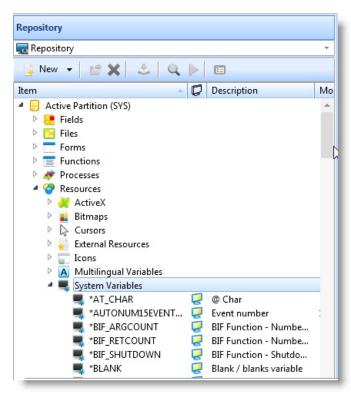
In this step, you will use *Refresh repository objects from the master system*. In Step 1. Create a LANSA for iSeries Object, you created a new System Variable that exists on the IBM i but is not yet known in Visual LANSA.

Note: If you are using Propagation, then your changes will be automatically synchronized in your Visual LANSA System.

1. Start Visual LANSA and logon.

You must logon using the same developer profile and task ID as used on LANSA for iSeries in order to ensure that you have authority to access the system variable created on the IBM i.

2. Using the Repository Brower, display a list of the current system variables. Your list will depend on how you have arranged the items on your Repository tab and may appear something like the following (shown undocked with Alphabetic listing off):

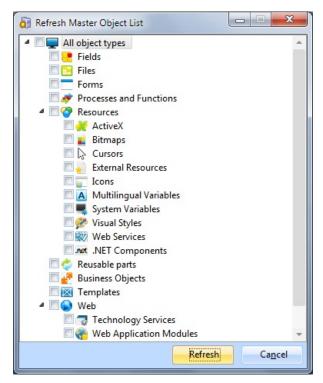


Your ***AUTOALP09iiiNUM** system variable will not be shown in this list.

3. Select the Maste Objects command in the ribbon.



The Refresh Master Object List dialog is displayed:

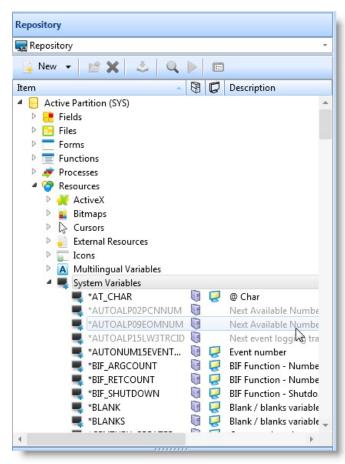


- 4. Select the System Variables to be refreshed.
- 5. Press the *Refresh* button.
- 6. The Check Out tab at the bottom of the editor will be displayed. Notice the status messages.

Wait for the refresh to be completed. The time required to complete the refresh will depend upon the number of system variables that are defined in the LANSA for iSeries Master System. You can speed up the repository refresh function by pressing F5. By default the repository refresh process runs more slowly as a background task, so you can continue development.

-						
×	Q	Job Status	Results	Description	Currently P	
무	~~	Completed	11415 found on master	Refresh system variables from master repository	2	
	×				]	
E.	X					
			Developmental De Chards Oak			
1.20	😇 Web Ancestor Elements 🛛 🧑 Breakpoints 🛛 😽 Check Out					
Refre	efreshing repository, 11055 objects pending ( F5 to refresh repository )					

7. Display the list of system variables. The *AUTOALP09iiiNUM should now appear.



Note that the *AUTOALP09iiiNUM system variable has an icon in its Master Repository Status column. Its Local Repository Status column is blank. This tells you that the definition of this system variable is currently not held in your Visual LANSA local repository. Objects which are not checked out to Visual LANSA are also shown in gray font.

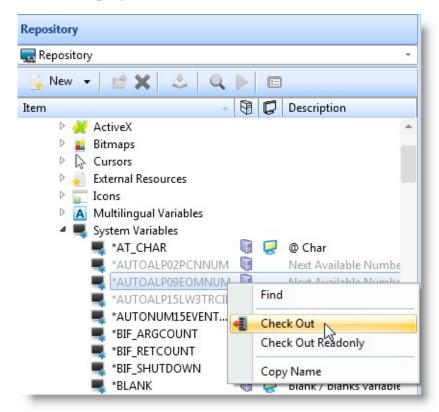
**Note:** If you are using Repository Synchronization, then changes made in the Repository on the IBM i or by any Visual LANSA developer in your PC Group will be automatically synchronized by checking out a read-only copy

of new and changed objects. Such changes will be reported in your Propagation tab. Refer to the *Visual LANSA Administrator's Guide* for further details of this feature.

# Step 3. Check Out Object

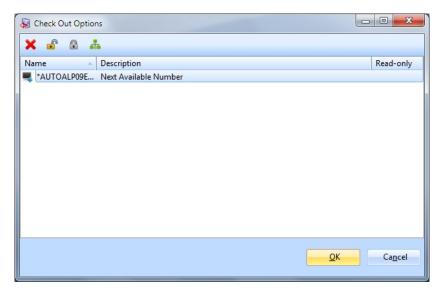
In this step, you will check out the system variable from LANSA for iSeries so that you can update the definition from Visual LANSA.

1. Using the Repository tab, select the *AUTOALP09iiiNUM system variable and right click to display the context menu.



2. Select the *Check out* option.

The *Check Out Options* dialog will be displayed.



- 3. Select the *AUTOALP09iiiNUM system variable. Notice that you can make the check out Read Only using the option buttons.
- 4. Press the *OK* button.
- 5. Display the Check Out tab at the bottom of the editor. Notice the status messages that are displayed.

×	P	Job Status	Results	Description	Currently Processing St
-		Completed	0 fatal errors - 0 warnings	*AUTOALP09EOMNUM - Next Available Number	21
1	×	Completed	223 found on master	Refresh system variables from master repository	21
	×	Completed	620 found on master	Refresh multilingual variables from master reposit	21
•		4			Þ
	Assist	ant   📩 Compile   📲 Chec	k In 🛛 📲 Check Out 🛛 📳 Propagation 🛛 👼	💂 Import   📋 Impact Analysis   🕜 Help	
				LANSA14Slave *S SYS VLXPGMLIB SET_E8 ENG Aud	t Courier Colors LANSA XHTML

- 6. Once the check out has been completed, use the Repository tab to open the *AUTOALP09iiiNUM system variable in the editor.
- 7. Change the system variable length from 7 to **9**.
- 8. Save and close the system variable.

# Step 4. Check In Changes

In this step, you will check in the changed system variable back to LANSA for iSeries. Once you have completed this step, you will delete the system variable from the IBM i.

- 1. Using the Repository tab, select the *AUTOALP09iiiNUM system variable and right click to display the context menu.
- 2. Select the *Check in* option.

Check in Options

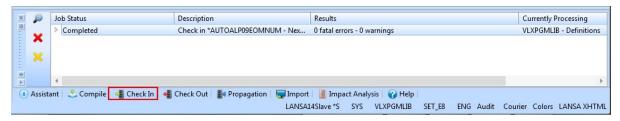
The *Check in Options* dialog will be displayed.

No check in options are required for system variables. If you were checking in an object, such as a form or function that can be compiled, a set of compile options would be displayed.

**Note:** If your task uses a "Release Locks" setting when checking in to the server, you will need to select the *Keep Locks* checkbox on this dialog, so that you can delete it from the repository in a later step.

3. Press the *OK* button to check in the changed system variable.

4. Display the Check In tab at the bottom of the editor. Notice the status messages that are displayed.



5. Once the check in has been completed, you may logon to LANSA for iSeries and view the *AUTOALP09iiiNUM system variable. The new definition should be displayed.

# **Step 5. Delete from Repository**

In this step, you will delete the *AUTOALP09iiiNUM system variable from the repository and specify that the system variable should also be deleted from the host.

1. Locate the *AUTOALP09iiiNUM system variable in the Repository tab and right click to display the context menu. Select the *Cross References* option to check if there are any dependencies before you delete the object.

Cross references - *AUTOALP09EOMNUM						
Cross References	Description	Qualifier	Reaso			
4 🜉 *AUTOALP09EOMNUM	Next Available Number					
Used by *AUTOALP09EOMN						
Uses *AUTOALP09EOMNUM						
4			•			
		Clos	se			

2. As there are no dependencies (in this case) close the dialog and select the *AUTOALP09iiiNUM system variable in the Repository tab and press the

X Delete icon in the Repository tab's toolbar.

3. The *Confirm delete component* dialog will be displayed. Select the *Delete from host option* so that the system variable is also deleted from the LANSA for iSeries Master Repository.

Confirm delete ob	-	X
	EO Next Available Number	Delete
		<u>C</u> ancel
Delete from host i	repository	

4. Display the *Propagation* tab at the bottom of the editor.

×	X	Sequen 🔻	Message	Date	Time	
<b></b>		7	SYSTEM VARIABLE named *AUTOALP09EOMNUM has been deleted successfully.	21/12/2012	2:56:11 PM	*
1	×	7	<==== (please note time) Host Repository Monitor STARTED in partition SYS, language ENG.	21/12/2012	9:23:55 AM	
		6 Request to delete Variable '*AUTOALP09EOMNUM' sent to the Host Repository.			2:56:11 PM	
-		6	<==== (please note time) Host Repository Monitor ENDED NORMALLY in partition SYS.	14/12/2012	3:25:19 AM	
•		5	End session request received by host monitor.	14/12/2012	3:25:19 AM	
	Inbound connection with system VLXPGMLIB successfully started, 21/12/2012 9:23:54 AM					*
	🕕 Assistant   👶 Compile   📲 Check In   📲 Check Out 🚦 Propagation 🐺 Import   📳 Impact Analysis   🤪 Help					
			LANSA14Slave *S SYS VLXPGM	LIB SET_E8	ENG Audit	Courier Colors LANSA XHTML

Notice the status messages are displayed.

## Things to Note

- You have just completed a simple exercise which demonstrates:
  - Refresh repository object from the Master
  - Check Out
  - Check In
  - Delete Object from Repository.

Refer to *Change Management* in the *Visual LANSA Administration Guide* for more information on these topics:

- Task Tracking Set up
- Using Task Tracking
- Repository Synchronization
- What are Repository Groups?
- What are Work Groups?

## Summary

#### **Important Observations**

- The Host Monitor will automatically be started when you perform a Check In or Check Out operation.
- When you Check Out objects, you can specify if the objects will be checked out as read only or for update. If checked out for update, you must have authority to the object.

# **Tips & Techniques**

• Review the use of Repository Synchronization (Propagation) is an efficient means of keeping your Visual LANSA systems current.

# What You Should Know

- How to refresh object lists from the LANSA for iSeries Master Repository.
- How to check out objects from the LANSA for iSeries Master Repository.
- How to check in objects to the LANSA for iSeries Master Repository.
- How to delete objects locally and in the LANSA for iSeries Master Repository.

# Visual LANSA Framework (VLF) Introduction

These exercises are to introduce you to the VISUAL LANSA Framework. The first two exercises are common to all Visual LANSA Framework users, and with the final three exercises, you will branch into either Web or Windows development.

With the WAM and Windows specific development you will start implementing real filters and command handlers, so you need to know how the data you will be using is stored.

The WAM and Windows specific exercises are based on the PSLMST Personnel demonstration file. Locate this file in the repository and view its properties. You can also see the layout of these files in the Appendix of the course notes.

An example of the Personnel System File:

Repository			Rules and Triggers Access Routes Bat		1		-
Repository	*	Field Name	Description	Ref. Field	Туре	Length	Dee
🖬 🗙 🗠 🔍 🕨 📼		<ul> <li>Primary Keys</li> <li>Primary Keys</li> <li>EMPNO</li> </ul>	Employee Number		Alpha	5	
ltem	- 🛃 Description	▲ ▼ Real Fields					
Þ 🔁 J	*	🔻 🔑 EMPNO	Employee Number		Alpha	5	100
Þ 🔚 К		SURNAME	Employee Surname		Alpha	20	3
E		GIVENAME	Employee Given Name(s)		Alpha	20	
M		ADDRESS1	Street No and Name		Alpha	25	
N		ADDRESS2	Suburb or Town		Alpha	25	
Þ 🔁 O		ADDRESS3	State and Country		Alpha	25	
4 🔁 P		POSTCODE	Post / Zip Code		Signed	6	
PSLEVENT	<ul> <li>Personnel Even</li> </ul>	PHONEHME	Home Phone Number		Alpha	15	-
PSLIMG	<ul> <li>Personnel Imag</li> </ul>	PHONEBUS	Business Phone Number		Alpha	15	
PSLMST	V Personnel	STARTDTER	Start date (YYMMDD)		Signed	6	
PSLSKL	<ul> <li>Personnel skills</li> </ul>	TERMDATER	Termination Date (YYMMDD)		Signed	6	
PSLTIMES	<ul> <li>Personnel time</li> </ul>	DEPTMENT	Department Code		Alpha	4	
Q		SECTION	Section Code		Alpha	2	
R		SALARY	Employee Salary		Packed	11	2
S							(
▷ 🔁 T		PJFs Before					
Þ 🔁 U		4 Read Virtuals					
V		E STARTDTE	Start Date (DDMMYY)	RETDAT	Signed	6	
⊳ 🔁 W		TERMDATE	Termination Date (DDMMYY)	RETDAT	Signed	6	233
▷ 🔁 X		MNTHSAL PIE-After	Monthly Salary	SALARY	Packed	11	2

## **Common Exercises**

When prototyping your application, (as in exercise LVF060) you decide on your business objects based on an analysis of the tasks of the users of your application. At that point the database structure and the required output is not important.

Following the first two exercises:

- LVF040 Execute Framework Application and
- LVF060 Create a Prototype

you will branch to follow either the Web path using LANSA's Web Application Modules (commonly called WAMs) for application development or the Windows path.

With the WAM and Windows specific development you will start implementing real filters and command handlers, so you need to know how the data you will be using is stored.

The WAM and Windows specific exercises are based on the PSLMST Personnel demonstration file. Locate this file in the repository and view its properties. You can also see the layout of these files in the Appendix.

# LVF040 - Execute Framework Application

## Objectives

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

%	No 5000-0000-0000-000-000-000-000-000-000-	Organizations		- 🗆 ×
<u>File Edit View H</u> elp <u>W</u> indows	(Framework) (Administration)			
				Quick Find
New Reports Web Site About	Address Resources Organization	Bookings Charges Spool Files	New Window Queues	
Organizations				
x				x
🗄 🏫 Favorites	Name	Code / Id Address 1	Address 2 Ad	ddress 3 Phone Nu Zip Code Sta
Kerner Application     Granizations	A Sesources Jones,Shirl		Pymble NS	SW 798 0543 2001
Resources	David Dateide			SW 222 2222 2147
🗄 😳 Programming Techniques	Pattinson,		e, Punchbowl NS	SW 212 3569 2016
🕀 🍕 Administration	Woods,Bra			SW 789 4562 2030
	Douglas,A			SW 639 5188 2147
	McCully,Lisa	A1021 15 Baker Place,	Penshurst NS	SW 159 6845 2153 -
				•
E				x
	🍇 Resource : Details (Paul,Pat	rick-A1012)		
	Details Documents	events 🛛 🔟 Images 🛛 📎 Notes	imeSheets	
			∧ _Skills	]
	Basic Details Employee Number	A1012	Description	Grade Comment
			4GL Programming	P Passed
	Employee Surname	Paul	Administratn Part 1 Administratn Part 2	D Met requi M Good marks
	Employee Given Name(s)	Patrick	Computer Science De	eg P Best in dass
	Start Date	01/05/1986 -	History Degree Company Introductio	M Could be on P Could do
	Employee Salary	26,456.04	Keyboard Skills	P Met requi
			Management Course	
	Address		Management Course Marketing Course 1	3 F Poor marks D Met requi
	Street No and Name	6 Camilo Avenue	Other Degree Course	e D Excellent
	Suburb or Town	Seven Hills	Programmer Producti	iv P xxxxxxxxx
	State and Country	NSW	New	Save Delete
	Deat 17in Cada	5147	v (	
				Save
🖃 🗊 🗃 🏦 🏹 1		Messages Rea	edy Local ENG	JIVORY13 5/12/13 13:56 🥥

To achieve this objective, you will complete the following:

- Step 1. Execute the Visual LANSA Framework
- Step 2. Execute an Application
- Step 3. Using Filters to Find an Employee
- Step 4. Using Commands and Command Handlers

Summary

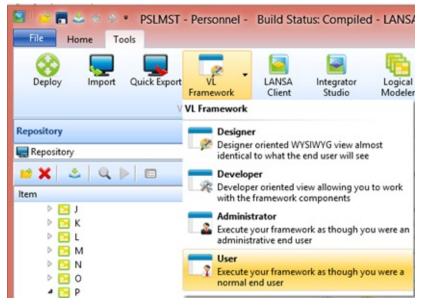
## **Before You Begin**

In order to complete this exercise, you must have completed the following:

• Check that you have met the prerequisites for the Visual LANSA Framework.

## Step 1. Execute the Visual LANSA Framework

- Start Visual LANSA.
   Log on to the DEM partition as follows:
   User ID: PCXUSER
   Password: PCXUSER
   Task ID: PCXTASK
   Partition: DEM
- 2. On the *Tools* ribbon, from the *VL Framework* menu , select the *User* option.



3. In the *Select Framework File* dialog select the option Open Latest Demonstration Version.

Select Framework File
JI_RAMP_TS.xml
JMI_Framework.xml
NL_SY001_SYSTEM.XML
vf_sy001_system.xml
VF_SY001_SYSTEM_LASTSHIPPED.XML
( 🔽 Open Latest Demonstration Version ) Open Cancel Browse

The Framework uses XML files to store the definition of your systems. The file

vf_sy001_system_lastshipped.xml always contains the latest demonstration system.

Note that if you only have one Framework file, this dialog is not displayed. The Framework window will appear.



## 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 left panel is the filter which is used to search through the employee data.

The right panel will show an instance list with the results of an employee search.

<b>D</b>	The essential business object	
<u>File Edit View H</u> elp <u>W</u> indo	WS	
	bout Address Resources Organization Bookings Charges Spool Files New Window Queues	uick Find
The essential business	s object	
Favorites     Favorites     Favorites     Formo Application     Programming Technique:     Basic     The essential busines     Auto filing the instan     Using the whole page     Showing a web page     Showing a web page     Showing a number page     Showing a busines	Specify a full or partial employee name. Employee Sumame S Filter Filter Number Name A1002 John Smythe A1005 Peter Smiths A1004 Ruth Smithson A0193 List List A1009 A0907 Robert Smithe	
Remembering values Passing information	Clear the current list of employees Search	Clear
Advanced     Prompting	The essential business object : Details (A0193-Fred Smithson)	_

## Step 3. Using Filters to Find an Employee

In this step, you will use a Filter to find employees. Filters allow you to search and sort the items in a business object. After an end-user has selected the employee business object, they typically want to locate a specific employee or list of employees.

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

2	The essential business object
<u>File Edit View H</u> elp <u>W</u> indov	vs
e - e	out Address Resources Organization Bookings Charges Spool Files New Window Queues
The essential business	object
Favorites     Favorites     HR Demo Application     Programming Technique     Basic     The essential busines     The CRUD business o     Auto filing the instan     Using the whole page     Showing a web page	Specify a full or partial employee name.     Number     Name       Employee Surname     B     A1404     Gillan Black       A3564     Freddy Brown     A0090     Fred Bloggs       A0070     Veronica Brown     A1031     John Blake
Selected, Current or ↓	Clear the current list of employees
Remembering values	The essential business object : Details (A0193-Fred Smithson)
Advanced	Employee Number A0193

## **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 the 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.

2	The ess	sential business object			- 🗆 🗙		
Eile         Edit         Yiew         Help         Window           Image: Comparison of the state of	õ 🖂 🤱	anization Bookings Charges Spool	l Files	Quick Find			
The essential business object							
Auto filing the instance	Specify a full or partial employee Employee Surname B	ze name.	Number A1404 A3564 A0090 A0070 A1031	Name Gilian Black Freddy Brown Fred Bloggs Veronica Brown John Blake			
Using the whole page Showing a web page Selected, Current or P Snap in Instance Lists	Clear the current list of employee Number	object : Details (A0070-Veronio	ca Brown)		Clear List		
Remembering values Passing information Advanced Prompting	Employee Surname Employee Given Name(s) Street No and Name	Brown Veronica 12 Railway Street			Save		
	Suburb or Town State and Country Post / Zip Code	Baulkham Hills NSW 2153					
< >	Home Phone Number	(02) 9609 4627 Messages Ready Loc	al ENG	JIVORY13	5/12/13 14:21 🍑		

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**

- The Visual LANSA Framework can be executed as a Visual LANSA form. Refer to VLF005 - Validating the Prototype.
- The Visual LANSA 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, the end-user can position the panels in the Framework or can float the panels as separate windows. These capabilities are part of the Framework and are not coded by the developer.
- The Visual LANSA Framework allows the end-user to perform actions in many different ways.
- Commands can be executed using menus, toolbar icons and pop-up menus.

# What I Should Know

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

## LVF060 - Create a Prototype

#### **Objectives**

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

4	_Employees -
File     Edit     View     Help     Windows       Image: New     Image: New     Image: New     Image: New     Image: New	(Framework) (Administration) Quick Find Address Resources Organization Bookings Charges Spool Files
_Employees	
x Favorites HR Demo Application HR Application Employees Statistical Reporting Programming Techniques Administration	Filter for _Employees.         This is a prototype of a filter program used to get         the _Employees to be displayed.         The user would normally enter search values here.         To see what a filter does, click on the "Emulate         Search" button.         Program Coding Assistant         Images Palette         Employee : Details (EMPLOYEE0005-Employee number 5)         Details         Address         Skills

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

Step 1. Understand the Requirements

Step 2. Create a Prototype Application – iii HR

Step 3. Define Filters and Command Handlers

Summary

## **Before You Begin**

• Complete exercise LVF040 – Execute a Framework Application

## **Step 1. Understand the Requirements**

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

- Two business objects, Employees and Statistical Reports.
  - Employees will be listed in the instance list based on searches By Name, By Start Date or By Location.
  - Employees will have actions, Details, Skills, Address and New.
- Statistical Reports will have actions Weekly Reports and Monthly Reports.

#### **Step 2. Create a Prototype Application – iii HR**

1. From the Tools ribbon, *VL Framework* menu, start Visual Frameworks using the Designer option.

To	ols						-
art	Quick Export	Framework	-	LANSA Client	lntegrator Studio	Logical Modele	
	Designer Designer oriented WYSIWYG view almost identical to what the end user will see				al View.		
		💦 Dev			ew allowing you omponents	ı to work	0

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.

Select Framework File	J
JI_RAMP_TS.xml JMI_Framework.xml NL_SY001_SYSTEM.XML vf_sy001_system.xml VF_SY001_SYSTEM_LASTSHIPPED.XML	
Prototype Mode Only     Open Latest Demonstration Version     Open     Cancel	Ī

The *Select Framework File* dialog is shown once you have opened any frameworks, other than the default vf_sy001_system.xml framework.

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

	LVF Workshop	
<u>File Edit View H</u> elp <u>W</u> indows	(Framework) (Administration)	
New Reports Web Site About	(New) (Properties) (Applications) (Commands) (Menus) (Design Code Tables)	pool Fi
<ul> <li></li></ul>	( Program Coding Assistant )	ore M
	(RAMP Tools ) (Virtual Clipboard )	-

4. Enter your new *Business Objects names*, **_Employees** and **Statistical Reports** separated by a comma. Notice the underscore character at the beginning of _Employees. This will avoid a conflict with the shipped framework Personnel Demonstration business objects.

Instant Prototyping Assistant	×
define business objects $\iff$ attach actions $\iff$ put business objects in groups $\iff$ generate prototype	-
This assistant will help you design a prototype in just a few minutes.	
Please remember that this is just an assistant to help you get started faster. Anything that you do with this assistant can be added to, changed or deleted later. You don't have to do the whole system in one go. You can do a subsystem first, and then come back later, to do more.	
The first step is to make a list of the names of "business objects" that the end users will work with.	E
We are not talking about database tables here or any other form of IT or OO "object" here. We just mean the everyday words that end users of the application use to describe the main "things" that they work with. For example: - Users dealing with an order processing application would use the words Orders, Customers and Products. - In working with e-mail people talk about Inboxes, OutBoy - A Human Resources application would use Employees - In many applications, users talk about the Daily Repo - A banking application would use Banks, Branches, Tre Step 1. Enter the names of the main "business ob, - Step 1. Enter the names of the main "business ob,	
Main Business Objects: Employees, Statistical Reporting	
Restore previous values	
Employees	
Next >>	Cancel

Click the *Next*>> button.

5. *Actions* will contain the default, Details, New and Notes actions. Define the additional actions required for _Employees and Statistical Reports. These are **Skills, Address, Weekly Reports** and **Monthly Reports**. Each should be separated by a comma.

Instant Prototyping Assistant				×
define business objects	ch actions	siness objects in groups	generate prototype	Î
What actions can users o	do with "busines	s objects" ?		
Step 2. Enter the names of all the act	tions below: (separated by	( commas)		
Windows designs use the <b>Object-&gt;Action</b> - (i.e. select the object you want to work the actions should be described in end use	with, and then choose what y	ou want to do with it.)		E
<ul> <li>Very concise words are used to describe "actions", because the object being worked with is already known.</li> <li>If you select an object in MS-Powerpoint and use the right mouse there is a concise menu option</li> <li>It does not say "Copy this text box to the Clipboard".</li> <li>Short verbs tend to be used to describe actions.</li> <li>(e.g. Copy, New, Edit, Print, Approve, Transfer, Reply, Renew )</li> <li>Short nouns are also used to refer to things that directly relate to the business object.</li> <li>(e.g. Details, History, Charges, Claims, Attachments, Schedule, Contacts, Documents, Experimentaria a "business object" called Customers, you could do these things to a "Edit, Print, Delete, Accounts, Recent Transactions, Correspondence, Verify .</li> </ul>				
Actions: Details , New , Notes , S	ikills, Address, Weekly Reports, Mor	thly Reports		
Step 3. Drag and drop the actions from the list below, onto all the appropriate business objects in the list on the right The same action can be used with many business objects.				
Details		🥔 _Employees 🍓 Statistical Reporting		*
<ul> <li>Notes</li> <li>Skills</li> <li>Address</li> <li>Weekly Reports</li> <li>Monthly Reports</li> </ul>				
			<< Back Next >>	Cancel

6. Hold down the *Control* key and select Detail, New, Skills and Address. Select the highlighted actions and hold down the left mouse to drag and drop them onto the **_Employees** business object.

Actions: Detai	ls , New , Notes , Skills, Address, Weekly Reports, Mont	nthly Reports	
Step 3. Drag and drop	Drag and drop Actions onto Business Object	oropriate business objects in the list on the right	*
The same action can be u			
Details New Notes Skills		_Employees Statistical Reporting	
Address Weekly Reports Monthly Reports			
		<< Back Next >> Cancel	

7. Repeat these steps to drag and drop actions Weekly Reports and Monthly Reports onto the **Statistical Reports** business object.

Your business object should now look like the following:



Click the *Next*>> button.

8. Using your initials instead of iii, define the iii HR application.

Drag and drop the _Employees and Statistical Reports business objects onto the **iii HR application**.

Instant Prototyping Assistant
define business objects attach actions + put business objects in groups + generate prototype
Grouping business objects
Step 4. Enter some application (/group) names below (separated by commas):
Business objects are grouped together to make security easier to manage and end user access simpler and more manageable. These types of business object groupings are named many different things (eg: Applications, Systems, Subsystems, etc).
For an example, an ERP system might have its business objects logically provided together into prolications or subsystems named : - Human Resources, Ordering, Dispatch, Accounts, Inventory, Production a Banking system might have its business objects logically grouped tog - Customer Accounts, Lending, Marketing . Banking system might have its business objects logically grouped tog - Customer Accounts, Lending, Marketing .
Don't organise business objects by user role - this can be done later.
Applications: Programming Techniques , Favorites , HR Demo Application , Administration , iii HR
Step 5. Now decide which application each business object belongs to.
Use drag and drop to put your bu Every business object must be put in Business Objects onto
Application  Application  Application  Application  Application  Application  Application  Application  (exists already)  (exists alr
<< Back Next >> Cancel

Your iii HR application should now look like the following:

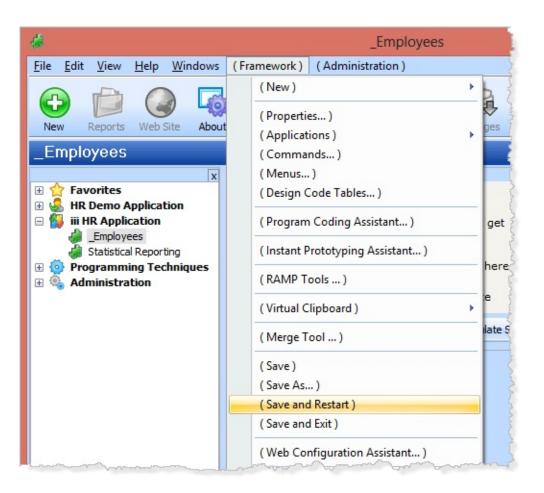
	<ul> <li>Programming Techniques</li> <li>Favorites</li> <li>HR Demo Application</li> <li>Administration</li> <li>iii HR</li> <li>Employees</li> <li>Statistical Reporting</li> </ul>	Application (exists already) Application (exists already) Application (exists already) Application (exists already) Application Business Object Business Object
	<< Back	Next >> Cancel

Click the *Next*>> button.

9. On the final dialog, click the *Finish* button to generate your **iii HR** application.

Instant Prototyping Assistant
define business objects $\iff$ attach actions $\iff$ put business objects in groups $\iff$ generate prototype The following will be generated when you press finish
A user of this framework sees 5 icons when they start the framework.
These icons represent 5 groups of business objects.
The first icon represents application ' Programming Techniques ' which contains business objects:
The second icon represents application ' Favorites ' which contains business objects:
The third icon represents application ' HR Demo Application ' which contains business objects:
The 4th icon represents application ' Administration ' which contains business objects: Creates the prototype
The 5th icon represents application ' iii HR ' which contains business objects: _Employees , Statistical Reporting
- When working with the _Employees business object, the user can do these things with a _Employee : Details , New , Skills
- When working with the Statistical Reporting business object, the user can do these things with a Statistical Reporting : Weekly Reports, Monthly Reputs
Kan

10. From the *Framework* menu, *Save and Restart* your framework.



Note: Your framework is an XML file. It is good practice to regularly save your work. You will also find that the Framework design tools will automatically prompt you to save your framework at regular intervals. By default this is every 10 minutes.

## **Step 3. Define Filters and Command Handlers**

In this step you will define three filters for _Employees and make a few basic enhancements to the prototype application. To enable you to quickly progress with these exercises, the enhancements steps have been kept to a minimum and the Statistical Reporting business object is not prototyped.

1. Open the Business Object Properties dialog for _Employees.

To do this select the _Employees business object on the Navigation panel and use the right mouse menu to select the Properties dialog.



2. Select the Icons tab:

😣 Business Object Pr				_		-	-		x
Identification Icons	Visual Styles	Filters	Filter Settings	Commands Enabled	Command Display	Custom Properties	SubTypes	Instance List / Relations	
				Select an icon t mployees busir object					
								Cl	ose

Select an icon for the _Employees business object.

3. Select the *Filters* tab. Change the Filter Name to By Name.

Business Object PropertiesE	mployees	_	
Identification Icons Visual Styles	Filters Filter Settings Commands	s Enabled   Command Display   Custom Properties   SubTy	ypes Instan
🔝 By Name	Identification Icons Filter Snap	o-in Settings	
	Caption	By Name The de	fault
	Hint:	filter	
	Sequence:	2	
	Internal Identifier:	F8B00611E2F1474CB2116532160995D1	2
	User Object Name / Type	F8B00611E2F1474CB2116532160995D1	Verify Nan
	hand		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

- 4. On the *Filters* tab, select the *Icons* tab and select any suitable icon.
- 5. Add a new filter by clicking on the New button.

8 Business Object PropertiesEr	nployees	
<ul> <li>Business Object PropertiesEr</li> <li>Identification Icons Visual Styles</li> <li>By Name</li> <li>By Start Date</li> </ul>	Filters Filter Settings Commands Identification Icons Filter Snap Caption Hint: Sequence: Internal Identifier: User Object Name / Type	Enabled Command Display Custom Properties SubTypes Instance List / Relations -in Settings By Start Date 3 50A902767AED47BA91166E88A84224A5 SoA902767AED47BA91166E88A84224A5 Verify Name
	<ul> <li>Allow on Web</li> <li>Allow in Windows</li> <li>Last Changed</li> <li>RAD-PAD File Name</li> </ul>	20110316-142543-JIVORY12 C:\PROGRA~2\LANSA\X_WIN95\X_LANSA\x_dem\execute_RADPAD_50A
New	Define new filter	Close

- a. Change the new filter's name to By Start Date and select any suitable icon.
- b. Click the New button and name the third filter By Location and select any suitable icon.
- 6. Select the Commands Enabled tab.
  - a. Select the New command in the Enabled column. Notice that it is a Business Object Command. This is because the command applies to the entire _Employees business object (instance list commands apply to the selected item in the instance list).

8 Business Object Pro	per	tiesEn	nployee	5		1
Identification Icons	Visua	al Styles	Filters	Filter Settings	S Commands Enabled	Command Dis
To enable and disable con lists	nmar	nds drag	them bet	tween these	🚱 New (NEW)	
Not Enabled	-	Enable	d		Choose Command	
About Framework		🖂 Add	ress		Business Object	Command
About		Det	tails		2	
Accounts		Nev	N	>	Sequence:	1
All Details		🦫 Skill	s		Constant Onlines	
All Entries	Ξ				-Command Options-	Defeate -
Amount			~~~	····	Stay Active	Default -

Next you specify how business object commands are displayed.

7. Select the Command Display tab.

	😪 Business Object PropertiesEmplo	/ees		
	Identification Icons Visual Styles Filt	ers   Filter Settings   Commands Enabled	Command Display Custom Properties	SubTypes
	Command Tab Style:	Tabs	<b>x</b>	
1	Command Tab Show All:	Automatic	×	ĺ
	Command Tab Location:	Тор	×	Č
1	Object Command Presentation	Separate stay on top window		2
	Instance Command Presentation	Use part of the window	×	)
	Multiline Tab Sheet Captions			
	Allow Float	ote		
	👻 The	edesign change you have just made may	Close	
	dov fully	ire your framework to be saved, shut in and then restarted before it becomes effective.	Save and Restart	
		rn me whenever I make this type of change.	······································	

a. For Object *Command Presentation*, select **Separate stay on top window**.

b. Click the *Save and Restart button* on the prompt that will appear. When your framework has restarted, it should look like this:

New Reports Web Site About	Address Resources Organization Bookings Charges Spool F	Quick Fin
_Employees		
Favorites     HR Demo Application     HR Application     in HR Application     Statistical Reporting     Programming Techniques     Administration	Image: Second start Date       Image: Second s	Employee Description
	Program Coding Assistant Images Palette Emulate Search	

8. With the _Employees business object selected, click on one of the ² New buttons. Notice that there is a New button in the main toolbar and in the

toolbar above the instance list. All these features may be configured differently.

The New dialog is displayed as a stay on top window. Notice that you can resize this window and use the Record Size button to remember the window size for the next execution. Note that the text can be edited.

- 9. Close the New dialog.
- 10. Click in one of your _Employees filter panels. Notice that these can also be edited. The panel provides a simple line editor. When adding text or images, type *Enter* to move to a new line as required.



11. Delete the existing text, add suitable text and drag and drop images from the images palette. Your objective is to make each panel "realistic" so that your users will understand the intended design when you review the prototype with them.

_Employees		
★       ★       Favorites         ★       ★       HR Demo Application         ★       ↓       ★         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓       ↓         ↓       ↓	By Name By Start Date By Location  Enter Employee Start Date :      28TH July 2003  Program Coding Assistant Images Palette Emulate Search	Employee Descriptio
	Images Palette	Drag & drop images onto panel

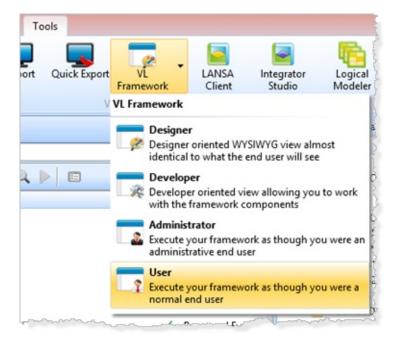
Restrict your changes to one filter panel. In your own projects, you would continue to enhance the appearance of every filter and command handler panel with suitable text and images.

- 12. Save and Restart your framework.
- 13. Display the properties of the _Employees business object and select the Instance List / Relations tab. Change the first two Column Captions to Number and Full Name. Note that there only two enabled columns, which have a column sequence number.

In your own applications you would probably enable additional columns and ensure that your filter components populate these extra columns.

8					Business Object P	roperties	Employee	s	
Identifical	tion Icons	Visual Styles	Filters Filt	er Settings   C	ommands Enabled   Con	mand Displa	y Custom Prope	rties SubTypes	Instance L
Do	while click for a	lefault commar	d		Save and Restore	Instance Li	the		
	ow multiple set				Enable Clear List				
-					-				
🖉 Ak	ow Instance L	ist to be sent t	o MS-Excel		File Prefix to be use	d for MS-Exc	el Sprei	dsheet_	(B
Popup	panel name			Enable	Pop Up Panels				
-	List Tool Bar I	eration		Top	rop op railos		-		
				and the second s					
Instance	List Tool Bar	Text Location		<none></none>			1		
Instance	List Tool Bar H	Height or Width	1	24					
Snap in 1	Instance List B	rowser							
Jequence	Type		Captor		dth % (Total 10	0 ⁴ Decimals	Edit Code	Date/Tin	ne Output P
10	VISUALID 1	Number			25		Default	SYSEMT8	
20	VISUALID2	Full Name			75		Default	SYSEMT8	
	ACOLUMN1					1	Default	SYSEMT8	
	ACOLUMN2						Default	SYSEMT8	
	ACOLUMN3						Default	SYSEMT8	
							Default	SYSEMT8	
	ACOLUMN4						Default	SYSEMT8	
=	ACOLUMN4 ACOLUMN5								
	ACOLUMNS	1							
This list c	ACOLUMINS ACOLUMIN6	ces of _Emplo	yees , and it	may also conta	in instances of	-	Default	SYSEMT8	

- 14. Close the Business Object Properties dialog.
- 15. Save and close your framework and re-open it as a User. Execute your prototype and ensure it meets the requirements.



Notice that the *Emulate Search* button will populate the instance list. 16. Select an instance list entry to display the _Employee command handler tabs.

x	x
Tavorites	🚱 By Name 🖉 By Start Date 📝 By Location 🖂 🗖 🚱 🍤
HR Demo Application HR Demo Application	Enter Employee Start Date : Number Full Name
Se _Employees	EMPLOYEmployee number
ightarresidential Statistical Reporting	28TH July 2003 EMPLOYEmployee number
Programming Technique:	
	_EMPLOYEmployee number
	Next >>EMPLOYEmployee number
	Emulate SearchEMPLOYEmployee number
	🙀 _Employee : Details (_EMPLOYEE0002Employee number 2)
	Details 🖂 Address 🦻 Skills

**Important note:** In order to limit the time taken, the prototyping section of the tutorials is brief. However, when you are creating a real Framework application, prototyping is the crucial step that will determine whether your project will succeed or fail.

A well thought out prototype will clearly communicate what the finished application will be like. You need to have it reviewed and signed off by both end-user representatives and developers. Users will know what they are getting and developers will know what they need to deliver.

#### Summary

#### **Important Observations**

- To create new applications 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. You can also create or extend an existing application using the Instant Prototyping Assistant.
- Applications can contain many business objects. Business objects are the objects that end-users work within the application.
- Filters enable end-users to search for business objects. A business object may have a number of filters.
- Command handlers enable end-users to carry out actions on business objects. Business objects may have many command handlers.

# **Tips & Techniques**

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

# What You Should Know

- How to create an application prototype using the Instant Prototyping Assistant.
- How to refine the application design with icons and additional filters.
- How to tune the behavior of command handlers.
- How to refine the appearance of the application filters and command handlers using text and the image palette.

## VLF for Web Application Module (WAM) Applications

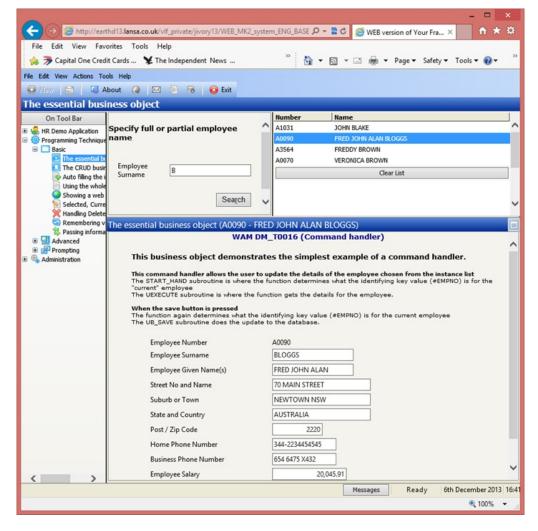
#### Applies to Web only using WAMs.

After you have created and validated your prototype, you can develop it into a functional application.

The basic structure and presentation of the application will remain unchanged as you continue to use the Framework. To complete the application, you simply replace the prototype filters and command handlers with real ones using WAMs.

In the following exercises, you will replace the employee filters with real filters and the Details prototype command handler with a real command handler:

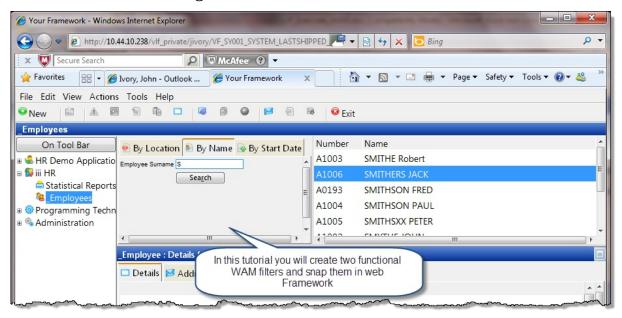
- LVF070WAM Snap in a Real WAM Web Filter
- LVF075WAM Snap in a Real WAM Web Command Handler
- LVF080WAM Add Instance List Columns in WAM Applications



## LVF070WAM - Snap in a Real WAM Web Filter

## Objective

• Learn how to replace prototype filters with real WAM filters. These will perform the actual selection of the items for the Instance List when the Framework is running in web mode.



To achieve this objective, you will complete the following:

- Step 1. Create Your Real WAM Filter
- Step 2. Snap In the WAM By Name Filter
- Step 3. Create a WAM By Location Filter
- Step 4. Snap in the WAM By Location Filter

## **Before You Begin**

You must have completed the preceding exercises.

You may wish to review the Visual LANSA Frameworks guide:

• Filters in the Key Concepts section.

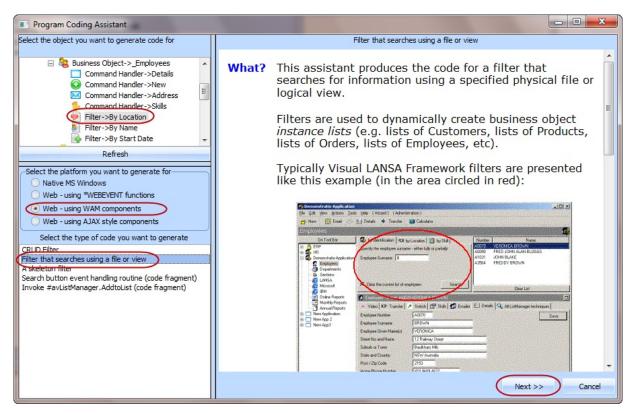
## Step 1. Create Your Real WAM Filter

In this step, you will create your own filter by creating a WAM which will be snapped into the Visual LANSA Framework.

- 1. Make sure that the *Enable Framework for WAMs* option in the *Framework Details* tab is selected. In the Visual LANSA editor check that you can create new WAMs (the option is available if your system is enabled for Web).
- 2. Start the *Program Coding Assistant* in the Framework using the *Framework* menu. The *Program Coding Assistant* window is displayed.

The *Program Coding Assistant* window allows you to create different types of components that can be plugged into your filters, instance lists and command handlers. It is strongly recommended that you use the *Program Coding Assistant* when you first start using the Framework. Initially you will most likely use filters that generate a component that can be executed e.g. *CRUD Filter* (Create/Read/Update/Delete), *Filter that searches a file or view*. As you progress you might only use a skeleton filter or simply copy from one that is similar to one that you want to create.

- 3. If you are using a non-English system, click on *Framework* then *Your Framework* in the top-left tree view. The *Set LANSA code generation preferences* option appears at the bottom. Select this option and set your preferences.
- 4. Select the iii HR application and then the *By Name* filter.
- 5. Choose *Web using WAM components* as the platform and *Filter that searches using a file or a view*.

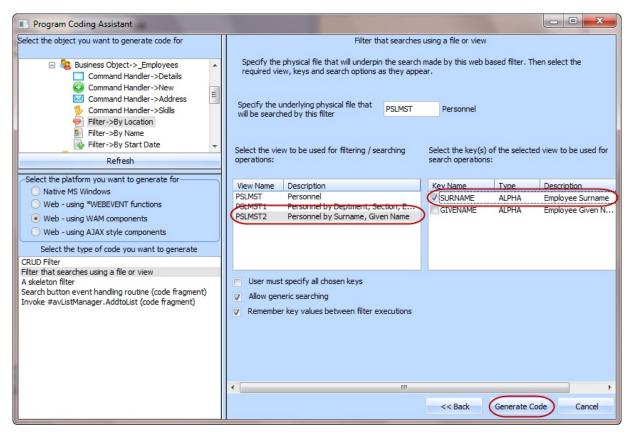


- 6. Click the *Next*>> button.
- 7. Specify PSLMST as the Physical File that most closely resembles this business object

Program Coding Assistant							
Select the object you want to generate code for	Filter that searches using a file or view Specify the identification protocol you have decided to use for this business object. If a physical file resembles this business object specify its name and the assistant will attempt to automatically deduce a basic identification protocol for you.						
Command Handler->Details Command Handler->New Command Handler->Address Command Handler->Skills	The	physical file that mbles this busine	most closely ss object is:	PSLMST Personnel			
Filter->By Name		Field Name	Type	Description	Drop Selected		
🚽 Filter->By Start Date 🔫	1	EMPNO	ALPHA	Employee Number	- Drop All		
Refresh	2	SURNAME	ALPHA	Employee Surname			
Select the platform you want to generate for       3 GIVENAME       ALPHA       Employee Given Name(s)         Native MS Windows       4       -       -         Web - using *WEBEVENT functions       -       Add fields from this Physical File       Add Keys       Add All							
Web - using WAM components     Web - using AJAX style components     Select the type of code you want to generate     PROGRAMMATIC IDENTIFIERS (for building AKey 1, 2, 3, 4, 5 and NKey 1, 2, 3, 4, 5 values)							
CRUD Filter		Field Name EMPNO	Type ALPHA	Description Employee Number	Drop Selected		
Filter that searches using a file or view A skeleton filter			ALPHA	Employee Number	E Drop All		
Search button event handling routine (code fragment) Invoke #avListManager.AddtoList (code fragment)	3				-		
	button filter						
	1	Field Name	Туре	Description	Drop Selected		
				<< Back	Next >> Cancel		

The *Program Coding Assistant* detects the Visual and Programmatic Identifiers required.

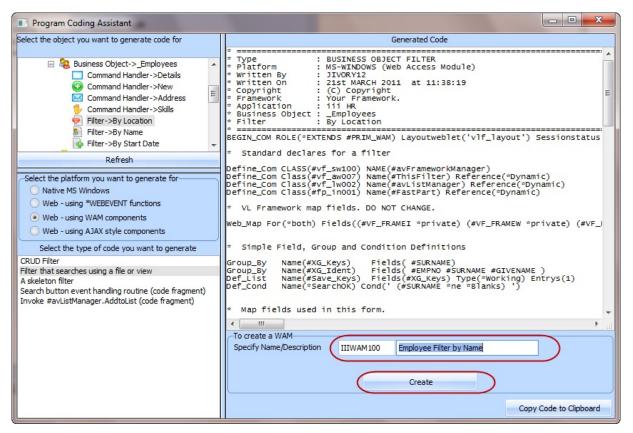
- 8. Click the *Next*>> button.
- 9. On the next page specify PSLMST2 as the view to be used for searching/filtering operations.
- 10. Specify **SURNAME** as the Key of the selected view to be used for search operations.



11. Click the *Generate Code* button.

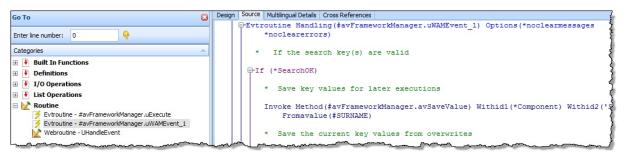
The next page, *Generated Code*, displays the source code for your filter. You now need to create the component that will contain this code:

- 12. Specify iiiWAM100 as the name of your real filter and By Name Filter as the description. (iii are your initials If you are using an unlicensed or trial version of Visual LANSA, you must always use the 3 characters DEM to replace iii).
- 13. Click the *Create* button to create the WAM.



After a brief delay a message is shown indicating the WAM has been created.

- 14. Switch to the Visual LANSA editor. Your filter is displayed in the Visual LANSA editor.
- 15. Use the GoTo tab to find the event routine #avFrameworkManager.uWAMEvent_1



Examine the code:

This statement tells the Framework that new entries are about to be added to the instance list:

Invoke Method(#avListManager.BeginListUpdate)

This statement clears the instance list:

Invoke Method(#avListManager.ClearList)

This statement reads the records that match the surname entered by the user: Select Fields(#XG_Ident) From_File(PSLMST2) With_key(#XG_Keys) Generic(*yes) Nbr_Keys(*Compute)

This statement sets up the visual Identifier(s) #UF_VisID1 := #EMPNO #UF_VisID2 := #SURNAME Use BConcat (#UF_VisID2 #GIVENAME) (#UF_VisID2)

This statement adds the data to the instance list

Invoke #avListManager.AddtoList Visualid1(#UF_VisID1) Visualid2(#UF_VisID2) AKey1(#EMPNO)

**VisualId1** will be shown in column one of the instance list and **VisualId2** will be shown in column two of the instance list. Akey1 is the key that uniquely identifies an employee (in this case the field is alphanumeric, so it's Akey1, not Nkey1).

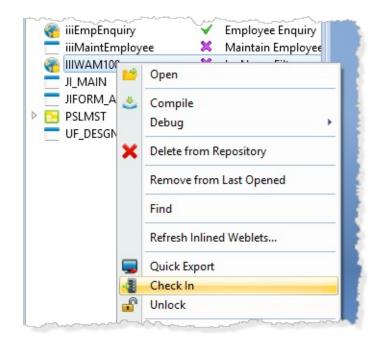
This statement tells the Framework that you have finished adding entries to the instance list:

Invoke Method(#avListManager.EndListUpdate)

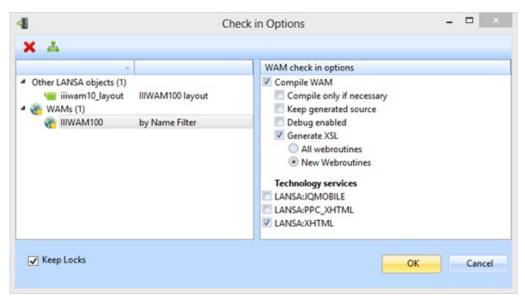
You may want to read *WAM Filter and Command Handler Anatomy* in the *Visual LANSA Framework Guide* to learn more about how WAMs are structured.

16. Select the *Compile* option on the *Home* ribbon.

- 17. Ensure the compilation was successful.
- 18. Close your WAM in the editor.
- 19.Select your WAM on the *Favorites / Last Opened* tab. Right click and use the context menu to select *Check in*.



- 20.On the *Check in Options* dialog, expand the list of objects. Note that the WAM layout as well as the WAM has been selected for Check in.
  - a.Ensure that the *Compile WAM* option is selected.
  - b.The *Generate XSL* and *LANSA:XHTML* checkboxes should also be selected.
  - c.In the example shown below, *Task Tracking* is set up to release the lock against PC Name when checking in. If the *Keep Locks* option is selected, this PC will retain a lock, which would allow subsequent changes to be made.



- d. Click *OK* to check the changes in.
- e. Wait until the compiles have finished.
- Note that VLF WAMs use a common VLF layout, which already exists on the server.

#### Step 2. Snap In the WAM By Name Filter

Now that you have compiled your new filter and are ready to test it, you need to snap it into the Framework.

- 1. In the Framework, close the *Program Coding Assistant*.
- 2. Select the iii HR application and double-click the **Employee** business object.
- 3. On the resulting *Business Object Properties* dialog, click the *Filters* tab.
- 4. Select the *By Name* filter. You will replace the web mock up filter with your real filter
- 5. Click the *Filter Snap-in Settings Tab*.
- 6. Click the WAM property radio button in the *Web Browser* group box.
- 7. Use the search button Sto open the Find dialog. Enter a *Like Name* value to find your WAM and click the *Find* button.

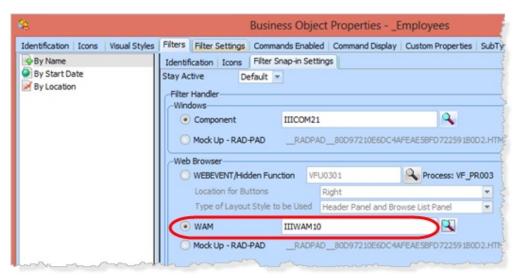


8.Select your By Name filter and click OK.

9	Find		×
Like Name	IIIWAM		
Like Description			
		Find	Refresh
Name	Description		
IIB-(AM070	Using a Tree View		
IIIWAM100	by Name Filter (IIIWAM10)		
IIWAM10 IIWAM110 IIWAM11 IIWAM120 IIWAM12	by Name Files Employee By Location Filter (IIIWAM11) Employee By Location Filter Employee Details command handler (IIIWAM12) Employee Details command handler		
		QK	Cancel

Important: Note that the *Description* includes the *Identifier* name in

brackets. The identifier must always be used for the component name in VLF.

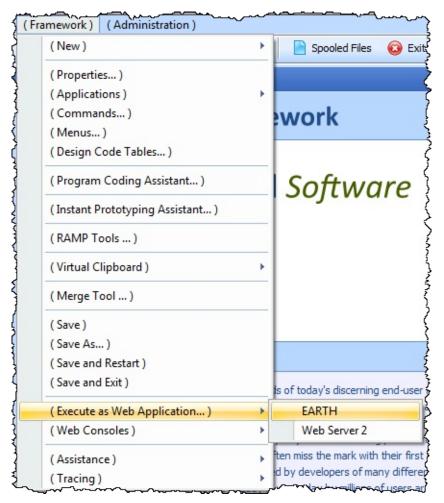


- 9. Click the *Instance List* tab.
  - a. Set the heading of the first column in the instance list to **Number** (it will display employee numbers) and the heading of the second column to **Name** (it will display employee names).
  - b. Deselect the Save and Restore Instance Lists option.

Business Object Prope	ertiesEmploy	ees			-	-	
dentification Icons Vis	ual Styles   Filter	s   Filter Setti	ings   Commands Enabled   Co	ommand Display	Custom Properties	SubTypes In	stance List / Relations
Enable Clear List Butt	on						
Double click for defau	lt command						
Save and Restore Ins	stance Lists						
Allow Instance List t	o be sent to MS-E	Excel	File Prefix to be used for MS-t	Excel S	preadsheet_	(ENG)	)
Instance List Tool Bar Location Top							
Instance List Tool Bar Height or Width 24		24					
Snap in Instance List Brow	ser						
Column Sequence	Column	Type	Column Caption	Width	n % (Total 70%)	Decim	als
10	VISUALID1		Number	10	1		Ŧ
20	VISUALID2		Name 60				ŧ
	ACOLUMN1						
	ACOLUMN2			·			
	ACOLUMN3		0				
	ACOLUMN4		0				
	ACOLUMN5		0				
	ACOLUMN6			-		-	

Note that the *Enable Clear List Button* is selected.

10. Use the (Framework) menu and select the option to save the Framework. Accept the prompt to upload the Framework and wait while the upload completes. 11. Use the (Framework) menu and select the option to *Execute as Web Application*...



- 12. Accept the default options and press OK.
- 13. Select the iii HR application in the web Framework and then the Employees business object. Bring the By Name filter topmost. Type in a partial surname and click Search.
- 14. Your filter is now snapped into the Framework and is usable.

C Your Framework - Windows Internet Explorer	
🚱 🔾 🗢 🙋 http://10.44.10.238/vlf_private/jivory/VF_SY001_SYSTEM_LASTSHIPPED_ 🖉 🖌 😣 49	🔁 Bing 🛛 🔎 👻
X 👿 Secure Search	
🖕 Favorites 🛛 👻 🏈 Ivory, John - Outlook 🧭 Your Framework 🛛 🗴 🛣 🖛 🖾 🔻 🖾	🖶 🔻 Page 🔻 Safety 🕶 Tools 👻 🔞 👻
File Edit View Actions Tools Help	
🔍 New 🖾 🖄 🖾 🖻 🖬 🗖 🖉 🖗 🔛 🙆 🐼 🚱 Exit	
_Employees	
On Tool Bar       By Location       By Name       By Start Date       Number       Name         Allous       Smithers Jack       Smithers Jack       Allous       Smithers Jack         Statistical Reports       Search       Allous       Smithers Jack         Programming Techn       Administration       Allous       Smithers Jack         Employees       Mumber       Name       Allous       Smithers Jack         Employees       Employees       Employee       Mumber       Allous       Smithers Jack         Employees       Employee       Employee       Employee       Employee       Employee       Employee         Employee       Employee       Employee       Employee       Employee       Employee         Employee       Employee       Employee       Employee       Employee       Employee         Employee       Employee       Employee       Employee       Employee       Employee       Employee         Employee       Employee       Employee       Employee       Employee       Employee         Employee       Employee       Employee       Employee       Employee       Employee         Employee       Employee       Employee       Employee       Employee <th>RED PAUL ETER</th>	RED PAUL ETER
🗖 Details 🐱 Address 🆻 Skills	
how when he was a second way we have the second sec	

## Step 3. Create a WAM By Location Filter

In this step, you will create a real By Location filter that will locate Employees by the department and section in which they work.

- 1. Start the *Program Coding Assistant*.
- 2. Select the Framework object navigation tree in the upper left of the *Program Coding Assistant* form.
- 3. Drill down through the tree to find your By Location filter and select it.
- 4. Choose Web using WAM components option as the platform.
- 5. Select the type Filter that searches using a file or view.
- 6. Click *Next*>>.

The *Program Coding Assistant* shows the PSLMST file as the physical file and detects the Visual and Programmatic Identifiers required. You do not need to change any of these values.

7. Click the *Next*>> button.

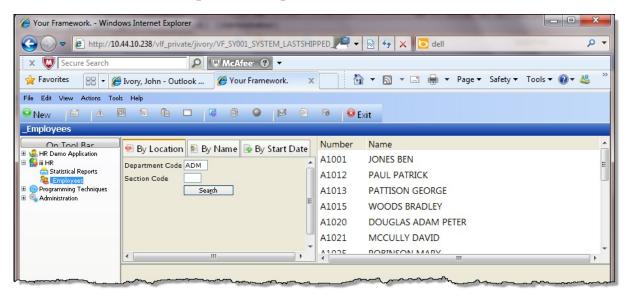
On the screen:

- a. Select the file view named PSLMST1 (Personnel by Department, Section, Employee Number).
- b. Select the search keys DEPTMENT and SECTION.
- c. Deselect User must specify all Chosen Keys.
- d. Deselect Allow Generic Searching.
- e. Select Remember key values between filter executions.
- 8. Click the *Generate Code* button. The right hand side of the *Program Coding Assistant* now shows the code that it has generated for your filter.
- 9. In the *Generated Code* window specify iiiWAM110 (where iii are your initials) as the name of your filter and give it a description. If you are using an unlicensed or trial version of Visual LANSA, you must always use the 3 characters DEM to replace iii.
- 10. Then click the *Create* button to create your filter.
- 11. Switch to the Visual LANSA editor. Compile the By Location WAM and choose to generate XSL for all web routines.
- 12. Check in and compile the WAM to the server.

## **Step 4. Snap in the WAM By Location Filter**

In this step, you will snap in your *By Location* filter.

- 1. In the Framework, close the *Program Coding Assistant*.
- 2. Select the iii HR application and double-click the Employees business object.
- 3. On the resulting Business Object Properties dialog, select the Filters tab.
- 4. Select the *By Location* filter. You will replace the web mock up filter with your real filter.
- 5. Click the *Filter Snap-in Settings* tab.
- 6. Click the WAM property radio button in the Web Browser group box.
- 7. As before use the *Search* button and the *Find* dialog to find your *By Location* WAM filter and return the *Identifier* as the component name..
- 8. Use the (*Framework*) menu and select the option to save the Framework.
  - a. Accept the prompt to upload the Framework and wait while the upload completes.
  - b. Use the *(Framework)* menu and select the option to *Execute as Web Application...*
  - c. Take the default options and press *OK*.



9. Your filter is now snapped into the Framework and is usable.

- 10.Test your By Location filter:
  - a.Enter a department code (e.g. ADM) and search to populate the instance list with employees for all sections in the department.
  - b.Enter a department code and section code (e.g. ADM and 01) to populate the instance list with employees for section 01 only.

#### Summary

#### **Important Observations**

• With snap-in real filters, you have now created real functionality in your application.

# **Tips & Techniques**

• The source code for the filters used in the demonstration application can be found in the repository in components named DM_FILT*.

# What I Should Know

- What you need to do to create your own WAM filters.
- How you snap them in the Framework.
- How to use the *Program Coding Assistant*.
- How to customize the way that instance lists are displayed.

# LVF075WAM - Snap in a Real WAM Web Command Handler

## Objective

- Learn how to replace prototype command handlers with real web handlers that will perform actual processing when the Framework runs in web mode.
- To replace the Details prototype command handler with a real WAM command handler.

Your Framework - Windo	ows Internet Explorer					- • ×		
	.44.10.238/vlf_private/jivory/	/F_SY001_SYSTEM_LASTSHIP	PPED PE	🖄 🍫 🗙 🔽 dell		+ م		
🗙 👿 Secure Search	2	McAfee 🕐 🔻						
🖕 Favorites 🛛 🖶 👻 後	Ivory, John - Outlook	🏉 Your Framework 🛛 🗙		🔻 🔝 👻 🖃 🖶 🕶 Page	e ▼ Safety ▼ To	ols 🔻 🔞 🕈 🚢 👋		
File Edit View Actions Too	ls Help							
🙆 New  🛋 🔅	🖻 🖻 🖻 🎑	A A A A A A A A A A A A A A A A A A A	🗟 🙆 E	xit				
_Employees								
On Tool Bar	🖲 By Location 🔊 By N	lame 😼 By Start Date	Number	Name	Department	Salary		
B Si ii HR Department Code ADM			A1001	JONES BEN	ADM	2,345.82 😑		
Statistical Reports	Statistical Reports		A1012	PAUL PATRICK	ADM	26,456.04		
<ul> <li>Programming Techniques</li> <li>Administration</li> </ul>	Search		A1013	PATTISON GEORGE	ADM	78,977.04		
			A1015	WOODS BRADLEY	ADM	313,000.04		
			A1020	DOUGLAS ADAM PETER	ADM	121,500.04		
			A1021	MCCULLY DAVID	ADM	87.000.04 -		
	_Employee : Details (A1001 - JONES BEN)							
	🗖 Details 🗾 Address	🐤 Skills						
		Save				<u>^</u>		
	Employee Number	A1001						
	Employee Surname	JONES				E		
	Employee Given Name(s) Street No and Name	BEN 144 Frog				E		
	Suburb or dept	PYMBLE.						
	State and Country	NSW.						
	Post / Zip Code	2001						
	Home Phone Number	799 5268						
	•		1			•		
						March 2011 15:15		
Done		<b>F</b>	🔞 😌 Ir	ternet   Protected Mode: Off	4	▼ € 85% ▼		

To achieve this objective, you will complete the following:

- Step 1. Create Your Real WAM Command Handler
- Step 2. Snap in Your WAM Command Handler
- 10. Test your By Location filter:

## **Before You Begin**

You must have completed the preceding exercises.

You may wish to review:

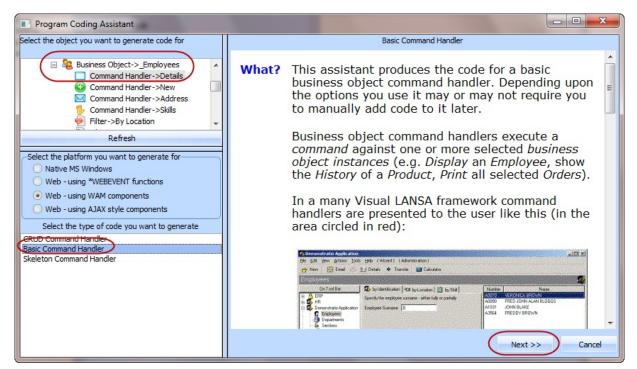
• Command Handler

• Framework Programming

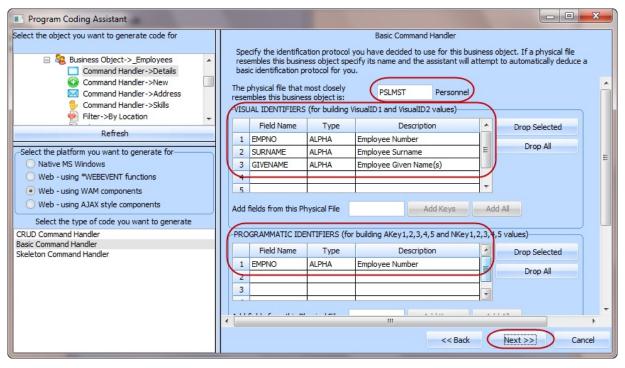
## Step 1. Create Your Real WAM Command Handler

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

- 1. Start the *Program Coding Assistant*.
- 2. Select the iii HR application, then the Details command handler.
- 3. Select Web using WAM components as the platform.
- 4. Select Basic Command Handler as the type of code you want create.

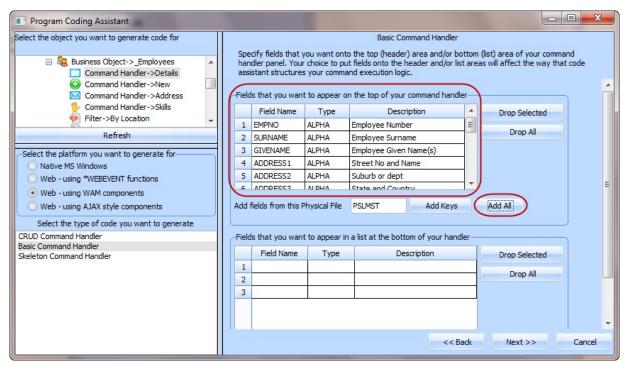


- 5. Click the *Next*>> button.
- 6. On the next page specify PSLMST as The physical file that most closely resembles this business object.



The *Program Coding Assistant* detects the Visual and Programmatic Identifiers required.

- 7. Click the *Next*>> button.
- 8. On the next page specify PSLMST in the field Add fields from this physical file in the section Fields that you want to appear at the top of your command handler.
- 9. Click the *Add All* button.

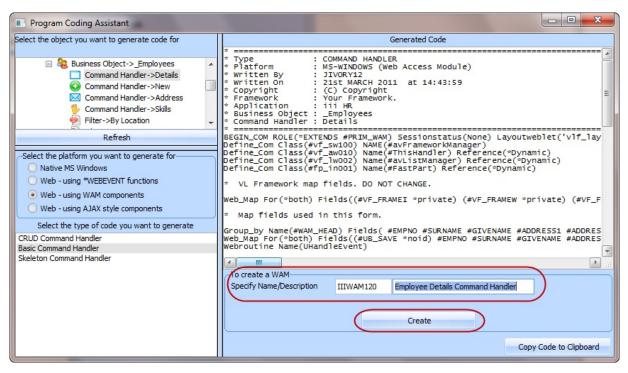


10. On the next page select the Include Default Save Button and Logic and click the Generate Code button.

The next page, *Generated Code*, displays the source code for your command handler.

You now need to create the component that will contain the code:

- a. Specify iiiWAM120 (where iii are your initials). Make the description of the component **Details command handler**.
- b. Click the *Create* button and wait until you see a message saying the component has been created in the development environment.



- 11. Switch to the Visual LANSA editor. The iiiWAM120 WAM is open in the editor.
- 12. Locate the #avFrameworkManager.uWAMEvent_1 handler and add a statement to save any changes made to the employee details. For example:

UPDATE FIELDS(#WAM_HEAD) IN_FILE(PSLMST) WITH_KEY(#EMPNO)

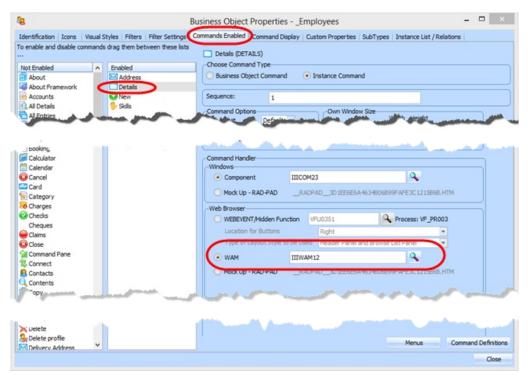
- 13. Locate the uInitialize event routine. This routine is always called when the command handler is invoked. Notice that it uses#avListManager.GetCurrentInstance method to get the key value of the currently selected item.
- 14. The uExecute event routine is only ever executed when the WAM is executed (that is, when a filter is started or a command handler is executed). When events occur inside an active WAM (for example a button click) uExecute is not signalled, just the registered uWAMEvent_N event.
- 15. Save the WAM.
- 16. Check in your changes to the server:
  - a. Compile your WAM locally.
  - b. If it compiles ok, select it in the *Repository* tab.

- c. Right-click the WAM to open the associated pop-up menu and choose the *Check in* option.
- d. In the *Check in Options* dialog select the option to generate XSL for all webroutines.
- e. Click *OK* to check the changes in.
- f. Wait until the compiles have finished.

# **Step 2. Snap in Your WAM Command Handler**

Once you have compiled your command handler and are ready to test it you need to snap it into the Framework. To snap in your own command handler:

- 1. Display the Framework.
- 2. Select the iii HR application and display the properties of the Employees object by double-clicking it.
- 3. On the resulting *Properties* dialog, click the *Commands Enabled* tab.
- 4. Select the *Details* command.
- 5. Click the *WAM* radio button in the *Web Browser* group box. Use the *Search* button to open the *Find* dialog. Locate your Details command handler WAM. Select it and click *OK* to return the *Identifier* name to the framework.



- 6. Use the *(Framework)* menu and select the option *Save the Framework*. Accept the prompt to upload the Framework and wait while the upload completes.
- 7. Use the *(Framework)* menu and use the option to *Execute as Web Application...* Select the default options and press *OK*.
- 8. Select the iii HR application in the web Framework and then the Employees business object. Bring the *By Name* filter topmost. Type in a partial surname

and click Search. Now click on an employee.

9. Your command handler for Details is now snapped in the web Framework and is usable.

🧭 Your Framework - Windo	ws Internet Explorer		x
🕒 🕞 🗢 🙋 http://10	.44.10.238/vlf_private/jivory/	VF_SY001_SYSTEM_LASTSHIPPED_🖉 🗸 🖄 😽 🗙 🔀 dell	<mark>ہ</mark> م
x 👿 Secure Search	2	V McAfee	
🔶 Favorites 🛛 🖶 🗸 🄏	Ivory, John - Outlook	🍘 Your Framework 🛛 🗙 🗸 🔝 🔻 🖃 🖶 🔻 Page 🔻 Safety 🔻 Tools 🕶 🔞 🕶 😂	,
File Edit View Actions Too			
		🖻 🕒 🖻 🗟 😰 Exit	
Employees			
On Tool Rar HR Demo Application Statistical Reports Composed Programming Techniques Administration	By Location By N Department Code ADM Section Code Search	A1001 JONES BEN A1012 PAUL PATRICK A1013 PATTISON GEORGE	
		+ A1015 WOODS BRADLEY	
	Employee : Details (A10 Control Details Address Employee Number Employee Surname Employee Given Name(s) Street No and Name Suburb or dept State and Country Post / Zip Code Home Phone Number Business Phone Number Department Code	Skills <u>Save</u> A1012           PAUL           PATRICK           6 Camillo Avenue           SEVEN HILLS.           NSW.           2147           687 1717           222 2222           ADM	
	•		F
		Messages Ready 21st March 2011	14:5
lone		Internet   Protected Mode: Off	-

- 10. Test your By Location filter:
  - a.Enter a department code (e.g. ADM) and search to populate the instance list with employees for all sections in the department.
  - b.Enter a department code and section code (e.g. ADM and 01) to populate the instance list with employees for section 01 only.

#### Summary

## What I Should Know

- What you need to do to create your own WAM command handlers.
- How you snap them in the web Framework.

# LVF080WAM - Add Instance List Columns in WAM Applications Objective

• Learn how to add columns to an Instance List in a WAM Framework application. In WAM browser applications, you can add columns to the shipped instance list. Specify the additional columns in the Instance List Settings tab sheet in the properties of the business object you are working with.

Ø Your Framework - Windo	ws Internet Explorer					
(3) ♥ (2) http://10	.44.10.238/vlf_private/jivory/	/F_SY001_SYSTEM_LASTSHI	PED PE	🗟 🏘 🗙 🔽 dell	-	• ۹
x 👿 Secure Search	2	McAfee				
🚖 Favorites 🛛 🖶 🗸 🄏	Ivory, John - Outlook	🖉 Your Framework 🛛 🗙		🔻 🔊 🔻 🖃 🖶 🔻 Page	e ▼ Safety ▼ To	ols 🔻 🔞 👻 🎽
File Edit View Actions Too	ls Help					
🛛 New 🗎 🛆 🖟		A A A A A A A A A A A A A A A A A A A	🗟 🔒 E	xit		
_Employees	4					
On Tool Bar	💌 By Location 🔊 By N	lame 😼 By Start Date	Number	Name	Department	Salary ^
🖻 👸 iii HR	Department Code ADM		A1001	JONES BEN	ADM	2,345.82 <mark>≡</mark>
Statistical Reports	Section Code		A1012	PAUL PATRICK	ADM	26,456.04
OProgramming Techniques     Administration	Search		A1013	PATTISON GEORGE	ADM	78,977.04
a			A1015	WOODS BRADLEY	ADM	313,000.04
			A1020	DOUGLAS ADAM PETER	ADM	121,500.04
			A1021	MCCULLY DAVID	ADM	87.000.04 -
	_Employee : Details (A10	01 - IONES BENI				
	Details Address	Skills				
	Details Prederess	Save				
	Employee Number	A1001				
	Employee Surname	JONES				=
	Employee Given Name(s)	BEN				
	Street No and Name	144 Frog				E
	Suburb or dept	PYMBLE.				
	State and Country Post / Zip Code	NSW.				
	Home Phone Number	799 5268				
	<		1	1		+
				Messages	Ready 21st	March 2011 15:15
Done			👩 🍚 Ir	ternet   Protected Mode: Off		
		~		, rotected model off		

Note: in this exercise, you will modify the By location filter. Normally, you should do the same modifications to the By name filter.

To achieve this objective, you will complete the following:

- Step 1. Add Columns to the Instance List
- Step 2. Change your filter
- Summary

#### **Before You Begin**

You must have completed the preceding exercises.

You may wish to review, in the Visual LANSA Framework User Guide:

- List Manager
- Adding Additional Columns to Instance Lists.

#### Step 1. Add Columns to the Instance List

In this step, you will configure your Employee business object to make the extra columns visible in the instance list.

- 1. Start the Framework as a designer.
- 2. Open the properties of the Employees business object.
- 3. Display the Instance List / Relations tab sheet.
- 4. Two visual identifiers are already defined.

Add two additional columns:

#### **Column Sequence Column Type Column Caption Decimals**

30	ACOLUMN1	Department	
30	ACOLUMINI	Department	

40 NCOLUMN1 Salary 2

B					<b>Business Object</b>	Properties -	_Employ	yees		
dentification	Icons V	isual Styles	Filters	Filter Settings	Commands Enabled	Command Disp	lay Custor	m Properties	SubTypes Instance List / Re	elations
Double	click for de	fault comman	đ		Save and Re	estore Instance I	Lists			
Allow	multiple selec	tions			Enable Clear	List Button				
Allow	Instance List	to be sent to	MS-Exce		File Prefix to be	e used for MS-Ex	cel	Spreadshee	et_ (ENG)	
Pop up par	el name			En	able Pop Up Panels					
Instance Lis	Tool Bar Lo	cation		Тор			*			
Instance Lis	Tool Bar Te	xt Location		<non< td=""><td>e&gt;</td><td></td><td></td><td></td><td></td><td></td></non<>	e>					
Instance Lis	Tool Bar He	ight or Width		24						
Snap in Inst	ance List Bro	wser					_			
Sequence	Type		Capti	on	Width % (Total 1	.00%) Decimal	s Ed	it Code	Date/Time Output Format	UTC
10	VISUALID1	Number			10		Default		SYSFMT8	Loca
22	VISUALID2	Full Name			60	_	Default		3131110	Loca
30	ACOLUMN1	Departmen	t		10		Default		SYSFMT8	Loca
40	NCOLUMN1	Salary			20	2	Default		SYSFMT8	Loca
	ACOLUMNIA	-					Defoult		SYSTMTO	Loca
	ACOLUMN3						Defeult		SYSEMT8	Loca

- Note:
  - Column widths may be set as a percentage.
  - Enable Clear List Button option has no effect in your own filters.
  - Numeric columns may have an edit code if needed.
  - Date columns may have an output format and UTC setting.

#### Step 2. Change your filter

Next, you need to make changes to your filter to fill the extra fields in the instance list with data.

- 1. Open the By Location filter iiiWAM110 which you created in *LVF070WAM Snapping in a Real WAM Web Filter*.
- 2. Make these changes to the code:
  - a. Change the GROUP_BY command to include the #DEPMENT and #SALARY field:

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

b. Locate Select Fields(#XG_Ident) command and change the AddtoList statement to set alpha column 1 and numeric column 1:

* Add instance details to the instance list Invoke #avListManager.AddtoList Visualid1(#EMPNO) AKey1(#EMPNO) Visualid2(#Surname) AColumn1(#Deptment) NColumn1(#Salary)

- 3. Compile your changed WAM.
- 4. If your Web server is on an IBM i, check in and compile your changed filter to the IBM i .
- 5. Restart the Framework and test the result.

🦉 Your Framework - Windo	ws Internet Explorer					
	.44.10.238/vlf_private/jivory/	VF_SY001_SYSTEM_LASTSHIP	PED 📮 🛨	🗟 <table-cell-rows> 🗙 🔽 dell</table-cell-rows>		۰ م
🗙 👿 Secure Search	2	McAfee 🕐 🔻				
🔶 Favorites 🛛 🖶 🗸 🄏	Vory, John - Outlook	🍯 Your Framework 🛛 🗙		🔻 🛐 🔻 🖃 🖶 🔻 Page	• ▼ Safety ▼ To	ols 🕶 🔞 🕶 🚢 炎
File Edit View Actions Too	ls Help					
🛛 New 🗎 🛆 🖟		🖻 🎱 💌 🖻	🗟 🙆 E	xit		
_Employees	1					
On Tool Bar     GRAME APPLICATION	💌 By Location 🔊 By N	lame 📑 By Start Date	Number	Name	Department	Salary
Fii HR Statistical Reports	Department Code ADM		A1001	JONES BEN	ADM	2,345.82 🗉
See Employees	Section Code		A1012	PAUL PATRICK	ADM	26,456.04
<ul> <li>Programming Techniques</li> <li>Administration</li> </ul>	Search	=	A1013	PATTISON GEORGE	ADM	78,977.04
			A1015	WOODS BRADLEY	ADM	313,000.04
			A1020	DOUGLAS ADAM PETER	ADM	121,500.04
	<	•	A1021	MCCULLY DAVID	ADM	87.000.04 -
	_Employee : Details (A10	001 - JONES BEN)				
	🗖 Details 🔀 Address	🐤 Skills				
		Save				<u>^</u>
	Employee Number	A1001				
	Employee Surname Employee Given Name(s)	JONES				=
	Street No and Name	144 Frog				Е
	Suburb or dept	PYMBLE.				
	State and Country	NSW.				
	Post / Zip Code	2001				
	Home Phone Number	799 5268				
	•				P 1 100 11	•
-					-	March 2011 15:15
Done		<b>J</b>	🛛 🚱 🚱 In	ternet   Protected Mode: Off	4	• • • 85% •

#### Summary

# What You Should Know

• How to add columns to an instance list in a browser WAM Framework application.

#### **VLF for Windows Applications**

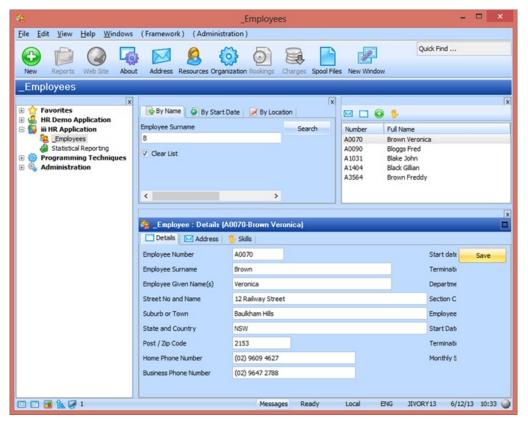
#### Applies to Windows only.

After you have created and validated your prototype, you can develop it into a functional Windows application in the following exercises:

- LVF070WIN Snap in Real Windows Filters
- LVF075WIN Snap in A Real Windows Command Handler
- LVF080WIN Add Instance List Columns in Windows Applications

The basic structure and presentation of the application will remain unchanged as you continue to use the Framework. To complete the application, you simply replace the prototype filters and command handlers with real Windows ones.

In these exercises, you will replace the employee filters with real filters and the Details prototype command handler with a real command handler:



#### **LVF070WIN - Snap in Real Windows Filters**

#### Objective

• Learn how to replace prototype filters with real filters which will perform the actual selection of the items for the Instance List.

<b>8</b>	_Employees -	×
<u>File Edit View H</u> elp <u>W</u> indows	(Framework) (Administration)	
	Quick Find	
New Reports Web Site About	Address Resources Organization Bookings Charges Spool Files	
_Employees		
Favorites     HR Demo Application     HR Application     Employees     Statistical Reporting     Programming Techniques     Administration	In this tutorial you will create two functional filters and snap them into the Framework	×

To achieve this objective, you will complete the following:

- Step 1. Create your Real By Name Filter
- Step 2. Snap in the By Name Filter
- Step 3. Filter Code
- Step 4. Create a Real By Location Filter
- Step 5. Snap in the By Location Filter
- Summary

#### **Before You Begin**

- You must have completed all the preceding exercises.
- You may wish to review in the Visual LANSA Frameworks Guide:
  - Filters in Key Concepts
  - Framework Programming

## Step 1. Create your Real By Name Filter

In this step, you will create a real filter which searches the PSLMST file by employee surname. You will also learn how to use the *Program Coding Assistant*.

1. Click the Program Coding Assistant button in the By Name filter.

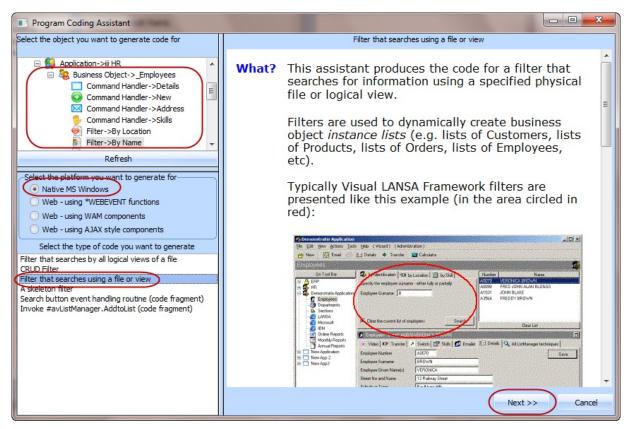
<b>6</b>	_Employees
<u>File Edit View H</u> elp <u>W</u> indows	(Framework) (Administration)
New Reports Web Site About	Address Resources Organization Bookings Charges Spool Files
_Employees	
★       Favorites         ★       ♣         HR Demo Application         ↓       ↓         ↓       ♣         ↓       ♣         ↓       ♣         ↓       ♣         ↓       ♣         ↓       ♠         ↓       ♠         ↓       ♠         ↓       ♠         ↓       ♠         ↓       ♠         Administration	By Name By Start Date   Enter Employee Start Date :   28TH July 2003   Next >>   Program Coding Assistant   Images Palette   Emulate Search

The *Program Coding Assistant* window is displayed. It allows you to create different types of components that can be plugged into your filters, instance lists and command handlers. It is strongly recommended that you use the *Program Coding Assistant* when you first start using the Framework.

Initially you will most likely use filters that generate a component that can be executed (e.g. Filter that searches by all logical views of a file, Filter that searches a file or view). As you progress you might only use a skeleton filter or simply copy from one that is similar to one that you want to create.

- 2. If you are using a non-English system, click on Framework / Your Framework in the top-left tree view. The Set LANSA code generation preferences option appears at the bottom. Select this option and set your preferences.
- 3. In the list on the top left, select the iii HR application and then the By Name filter.
- 4. Underneath it, select Windows as the platform. Entries in this list will depend on which platforms are enabled in your framework.

5. As the type of code you want to generate, select Filter that searches using a file or a view.



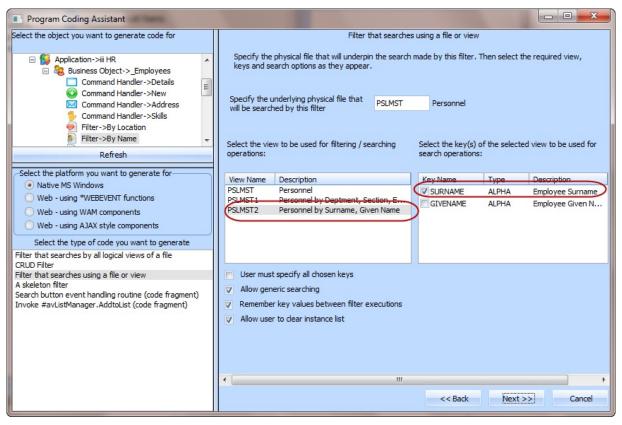
- 6. Click the *Next*>> button.
- 7. On the next page specify PSLMST as The physical file that most closely resembles this business object.

Program Coding Assistant							
Select the object you want to generate code for		500	cifu the identifica		ilter that searches using a file or vie you have decided to use for this bus		al filo
Application->iii HR     Application->iii HR     Command Handler->Details     Command Handler->New     Command Handler->Skills     Command Handler->Skills		rese basi The p reser	embles this busine c identification p ohysical file that i nbles this busines	ess object spec rotocol for you nost closely is object is:	cify its name and the assistant will a		
Filter->By Location			Field Name	Туре	Description	Drop Selecter	d
🗧 Filter->By Name 👻		1	EMPNO	ALPHA	Employee Number	- Drop All	
Refresh		2	SURNAME	ALPHA	Employee Surname		
		3	GIVENAME	ALPHA	Employee Given Name(s)		
Select the platform you want to generate for		4					
Web - using *WEBEVENT functions Web - using WAM components Web - using AJAX style components			fields from this Ph		Add Keys	Add All	E
Select the type of code you want to generate	11	PRO	GRAMMATIC IDE	NTIFIERS (for	building AKey1,2,3,4,5 and NKey1,	,2,3,4,5 values)	
Filter that searches by all logical views of a file			Field Name	Туре	Description	Drop Selecter	d d
CRUD Filter		1	EMPNO	ALPHA	Employee Number	E Drop All	
Filter that searches using a file or view A skeleton filter		2					
Search button event handling routine (code fragment)		3				+	
Invoke #avListManager.AddtoList (code fragment)			fields from this Ph		Add Keys	Add All	
	1	ADD	TTIONAL COLUM	NS (for building	g AColumn<> and NColumn<> value	es)	
			Field Name	Type	Description	Drop Selecter	d -
		1		1			
					<< Back	Next >>	Cancel

The *Program Coding Assistant* guesses the Visual and Programmatic Identifiers required:

- A *Visual Identifier* is the field or fields that a user would use to identify a unique instance of the business 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 represent the additional columns in your instance list that you may have added during the prototyping phase.
- 8. Click the *Next*>> button.
- 9. On the next page specify PSLMST2 as the view to be used for filtering/searching operations. It is logical view of the PSLMST file keyed by the SURNAME and GIVENAME fields.
- Note that you need an appropriate logical file for each filter that you want to create. Before implementing all your filters, review your data model to confirm that all the logical files exist. Doing so will speed up the process of implementing your prototype.

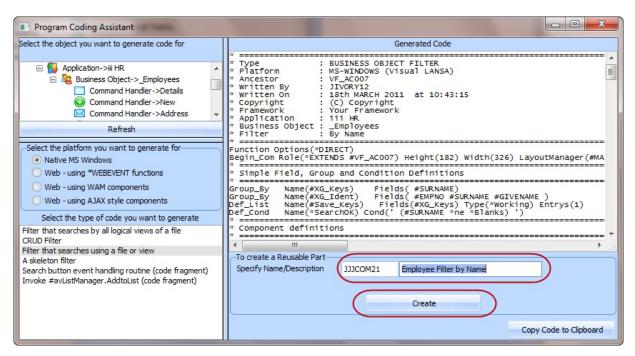
# 10.Select the SURNAME field as the key of the view to be used for search operations.



- 11.Click the *Next*>> button. Ignore the options on this page.
- 12.Click the *Next*>> button.
- 13.On the next page click the *Generate Code* button.

The *Generated Code page* displays the source code for your filter. You now need to create the component that will contain this code:

- 14.Specify iiiCOM21 as the name of your real filter and By Name Filter as the description. (iii are your initials If you are using an unlicensed or trial version of Visual LANSA, you must always use the 3 characters DEM to replace iii).
- 15.Click the *Create* button to create the component in Visual LANSA.



After a brief delay the message *Created in the development environment* is displayed.

-To create a Reusable Part			
Specify Name/Description	111COM21	Employee Filter by Name	
Created in the development er	nvironment		
		Create	
			Copy Code to Clipboard

16.Switch to the Visual LANSA editor and compile the component.

# Step 2. Snap in the By Name Filter

Now that you have compiled your new reusable component (filter) and are ready to test it, you need to snap it into the Framework.

- 1. In the Framework, close the *Program Coding Assistant*.
- 2. Double-click the Employees business object to display its properties.
- 3. Display the *Filter Snap-in Settings* tab.
- 4. Click the *Search* Sutton to display the *Find* dialog.

9	Find		×
Like Name	IIICOM		
Like Description			
		Find	Refresh
Name	Description		
IIICOM18	Employee Maintenance		
dIICOM21	Employee By Name Filter		
IIICOM22 IIICOM23	By Location Filter Details Command Handler		
		QK	

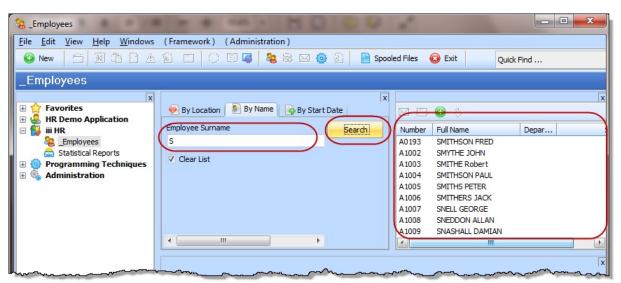
Enter a *Like Name* value to find your filter and click *Find*.

- Select your By Name filter and click *OK* to return the *Identifier* to the framework. Component names in the framework must always use an *Identifier*.
- In this case, the *Identifier* generated is the same as the short Name given to this filter.



5. Close the Employees business object properties and display the *By Name* filter. You can now see your real filter.

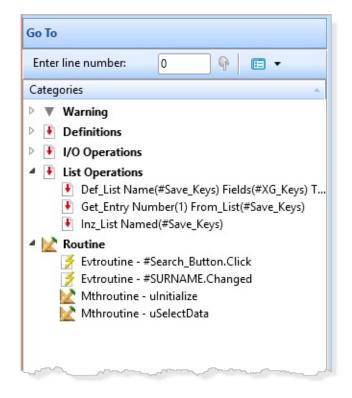
6. Type in a letter in the Surname field and click the Search button to verify that your real filter has been snapped in the Framework and is usable.



# **Step 3. Filter Code**

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

- 1. Switch to the Visual LANSA editor where the iiiCOM21 component is open.
- 2. Use the GoTo tab to select the **uSelectData** method routine.



Review the generated source code in the *Source* tab to see how the filter is coded to add data to the instance list:

The Framework is notified that an update is about to occur.

```
Invoke #avListManager.BeginListUpdate
```

Next, the list is cleared of any existing items.

Invoke #avListManager.ClearList

Next, data is selected. You can use one of the techniques you learnt in the Visual LANSA Fundamentals exercises to do this. For example:

Select Fields(#XG_Ident) From_File(PSLMST2) With_key(#XG_Keys) Generic(*yes) Nbr_Keys(*Compute) Next, the visual identifiers are set up: Change #UF_VisID1 #EMPNO Change #UF_VisID2 #SURNAME

Then the data is added to the list.

```
Invoke #avListManager.AddtoList Visualid1(#UF_VisID1)
Visualid2(#UF_VisID2)
AKey1(#EMPNO)
```

*VisualId1* will be shown in column one of the instance list and *VisualId2* will be shown in column two of the instance list. Akey1 is the key that uniquely identifies an employee (in this case the key field is alphanumeric).

Finally, the Framework is notified that the instance list update is complete. Invoke #avListManager.EndListUpdate)

3. Click on *Details* tab in the editor to display the properties of your component.

Details		
IIICOM21 - Employee By Nam	e Filter 🗸 👻	
Properties Events Method	5	
E • ?		
Ancestor	#VF_AC007	~
BusyUpdates	Wait	
BusyUpdatesOfParent	False	
ComponentClassName	IIICOM21	
🖻 ComponentPatternNam	IIICOM21	
🖻 ComponentTag		
🖻 ComponentTypeName	IIICOM21	
Cursor	*NULL	
DefaultProperty		
DisableNoScroll	False	
DisplayPosition	1	
🖉 DragStyle	None	
EnableChildren	False	

You need to ensure that all properties are displayed:

- 4. From the *File* menu, select *Options*.
- 5. In the LANSA Settings dialog, select the General group, click on Details and

make sure the Show Advanced Features option is selected.

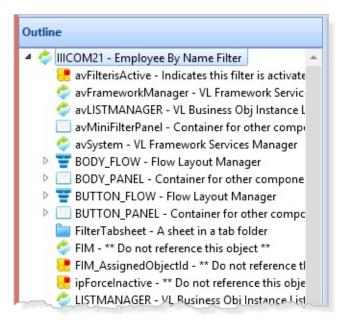


Click *OK* to close the *LANSA Settings* dialog.

6. Notice that the Ancestor property of the component is #VF_AC007. All filters inherit from this base class which provides a set of predefined behavior.

Details					
IIICOM21 - Employee By Name Filter 🔹 🗸 🧔					
Properties Events Methods					
<b>□</b> • ?					
Ancestor	#VF_AC007	^			
BusyUpdates	Wait				
BusyUpdatesOfParent	False				
ComponentClassName	IIICOM21				
ComponentPatternNam	IIICOM21				
ComponentTra	mon				

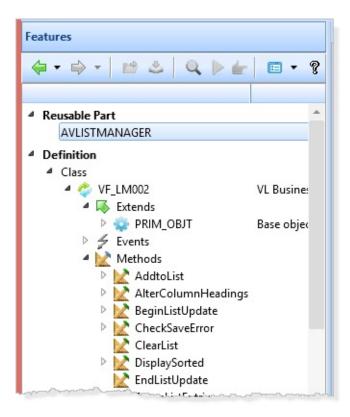
7. Click the *Outline* tab in the editor to see what components you inherit from the VF_AC007 ancestor component.



8. Right-click the avLISTMANAGER component and select the *Features* option.

			4					
🗇 IIICOM21 - Employee By Name Filter	-							
📒 avFilterisActive - Indicates this filter is activ	vate		_					
🗇 avFrameworkManager - VL Framework Ser	vic		* Type			BUSINESS	OBJECT	FTL
🗇 avLISTMANAGER - VL Business Obj In 👝	David	able Part:				Instance of VF		
avMiniFilterPanel - Container for othe	Reus	able Part:	AVEISTIM	ANAGER	-	instance of VP	_LIVIOU2	
🗇 avSystem - VL Framework Services Ma	Delet	e Compo	nent			Features		
BODY_FLOW - Flow Layout Manager	Copy	Compon	ent			Sth DECEN	MBER 201	3
BODY_PANEL - Container for other co		Compone				(C) Copys	right	
BUTTON_FLOW - Flow Layout Manag		compone			- :	LVF Works	shop	
BUTTON_PANEL - Container for other	Goto	Definition	1		1	iii HR Ap	oplicati	ion
🚞 FilterTabsheet - A sheet in a tab folder	Save	Definition			5 1	_Employee	88	
FIM - ** Do not reference this object *					- :	By Name		
FIM_AssignedObjectId - ** Do not refe 🗲	Event	ts: VF_LM	002		>			

9. Expand the methods of the component and examine them.



If you double click on any of the methods, help text is displayed in the *Help* tab.

10. Close the iiiCOM21 component.

You may want to read Windows Filter and Command Handler Anatomy in the in the Visual LANSA Framework Guide to see how these components are structured.

#### **Step 4. Create a Real By Location Filter**

In this step you create a real By Location filter.

- 1. In the Framework start the *Program Coding Assistant*.
- 2. Drill down through the tree to find your *By Location* filter and select it.
- 3. Select *Native MS Windows* as the platform to generate for and the type as *Filter that searches using a file or view*.
- 4. Press the *Next*>> button.
  - a. For your VISUAL IDENTIFIERS specify fields EMPNO, GIVENAME and SURNAME
  - b. For your PROGRAMATIC IDENTIFIERS specify field EMPNO only.
  - c. No ADDITIONAL COLUMNS should be specified.
- 5. Click the *Next*>> button to move the *Program Coding Assistant* forward to the next prompt. This prompt is asking you to select the file or view that the filter should use for searching.
  - a. PSLMST should already be specified as the Underlying Physical File.
  - b. Select the view named PSLMST1 (Personnel by Department, Section, Employee Number).
  - c. Select the search keys DEPTMENT and SECTION.
  - d. Deselect User must specify all Chosen Keys.
  - e. Deselect Allow Generic Searching.
  - f. Select Remember key values between filter executions.
  - g. Select Allow user to clear instance list.
- 6. A screen with additional options is displayed. Do not select any. Click the *Generate Code* button. The right hand side of the *Program Coding Assistant* now shows the code that it has generated for your filter.
- 7. In the *Generated Code* window, specify iiiCOM22 as the name of your new filter and give it a description.

Click the *Create* button to create your filter.

8. Switch to Visual LANSA and your filter should be open in the editor. Compile the new component.

# Step 5. Snap in the By Location Filter

In this step, you will snap in your By Location filter.

- 1. In the Framework close the *Program Coding Assistant*.
- 2. Select the iii HR application and double click the Employee business object.
- 3. On the resulting *Business Object Properties* dialog, select the *Filters* tab.
- 4. Select the *By Location* filter.
- 5. Select the *Filter Snap-in Settings* tab.
- 6. Use the *Search* button to display the *Find* dialog. Find and select your By Location filter and click *OK* to return the *Identifier* to the framework.
- 7. Your filter is now snapped into the Framework and is usable.
- 8. Close the business properties dialog and test the By Location filter.
  - a.Enter a full department code (e.g ADM) to populate the instance list with employees for all sections in this department.
  - b.Enter a department code and section code (e.g ADM and 01) to populate the instance list with employees for one section only.

#### Summary

#### **Important Observations**

• With snap-in real filters you have now created real functionality in your application.

# **Tips & Techniques**

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

# What I Should Know

- What you need to do to create your own filters.
- How you snap them in the Framework.
- How to use the *Program Coding Assistant*.

# LVF075WIN - Snap in A Real Windows Command Handler

#### Objective

- Learn how to replace prototype command handlers with real handlers which will perform actual processing.
- To replace the Details prototype command handler with a real command handler.

🍓 _Employee : Details (A	0907-Simpson Anne)
Details 🖂 Address	😕 Skills
Employee Number	A0907 Save
Employee Surname	Simpson
Employee Given Name(s)	Anne
Street No and Name	33 Anne street
Suburb or Town	Annevile
State and Country	NSW
Post / Zip Code	2145
Home Phone Number	090909
Business Phone Number	090909
Start date (YYMMDD)	94/03/03
Termination Date (YYMMDD)	0/00/00
Department Code	AUD
Section Code	In this tutorial you will create a real
Employee Salary	34,213.04 command handler for the Details
Start Date (DDMMYY)	030394 command and snap it into the
Termination Date (DDMMYY)	oooooo framework
Monthly Salary	2,851.09
	Messages Ready Local ENG JIVORY13 5/12/13 16:40

To achieve this objective, you will complete the following:

- Step 1. Create your Real Command Handler
- Step 2. Snap in your Command Handler
- Summary

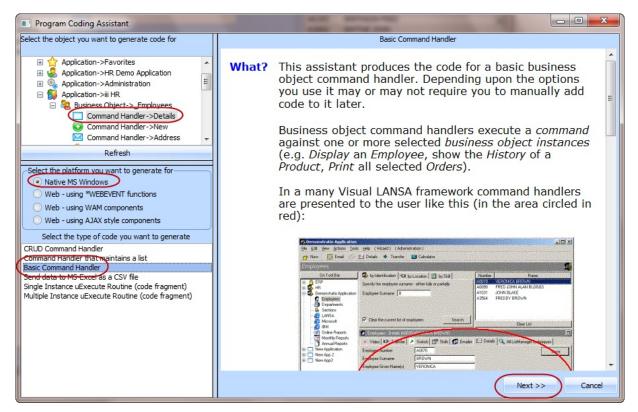
#### **Before You Begin**

- You must have completed all the preceding exercises.
- You may wish to review:
- Command in *Key Concepts* in the *Visual LANSA Framework Guide*.

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

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

- 1. Start the *Program Coding Assistant*.
- 2. In the list on the top left of the *Program Coding Assistant* window, select the iii HR application and then the Details command handler.
- 3. Select *Native MS Windows* as the platform and *Basic Command Handler* as the type of code.



The *Basic Command Handler* is the most commonly used assistant, as you typically want to create a command handler that displays your data. You then customize the page to meet your specifications.

The *CRUD Command Handler* is used in conjunction with the *CRUD filter* and only if the commands defined for the business object are *New*, *Details*, *Copy*, *Delete*.

The *Command Handler* that maintains a list allows you to generate code so that you can use just one command handler to view the details of the instance and a list of information about related data.

4. Click the *Next*>> button.

5. On the next page specify PSLMST as *The physical file that most closely resembles this business object*.

Program Coding Assistant		
Select the object you want to generate code for	Basic Command Handler	
Application->Favorites     Application->HR Demo Application     Application->HR Demo Application     Application->Administration     Application->iii HR     Business Object->_Employees     Command Handler->Details	Specify the identification protocol you have decided to use for this business object. If resembles this business object specify its name and the assistant will attempt to autor basic identification protocol for you. The physical file that most closely resembles this business object is:	
Command Handler->New	Field Name Type Description _ Dro	p Selected
Command Handler->Address 🚽	1 EMPNO ALPHA Employee Number	Drop All
Refresh	2 SURNAME ALPHA Employee Surname =	
Select the platform you want to generate for	3 GIVENAME ALPHA Employee Given Name(s)	
Native MS Windows	4	
Web - using *WEBEVENT functions Web - using WAM components Web - using AJAX style components	Add fields from this Physical File Add Keys Add All	E
Select the type of code you want to generate	PROGRAMMATIC IDENTIFIERS (for building AKey 1, 2, 3, 4, 5 and NKey 1, 2, 3, 4, 5 values)	)
CRUD Command Handler		p Selected
Command Handler that maintains a list	1 EMPNO ALPHA Employee Number	Drop All
Basic Command Handler Send data to MS-Excel as a CSV file		
Single Instance uExecute Routine (code fragment)	3	
Multiple Instance uExecute Routine (code fragment)	Add fields from this Physical File Add Keys Add All Add Keys Add Add Add Add Add Add Add Add Add Ad	
	Field Name Type Description Dro	p Selected
	< III	+
	<< Back	Next >> Cancel

The *Program Coding Assistant* guesses the Visual and Programmatic Identifiers required.

- 6. Click the *Next*>> button.
- 7. On the next page specify PSLMST in the field Add fields from this physical file in the section Fields that you want to appear at the top of your command handler.
- 8. Click the *Add All* button.

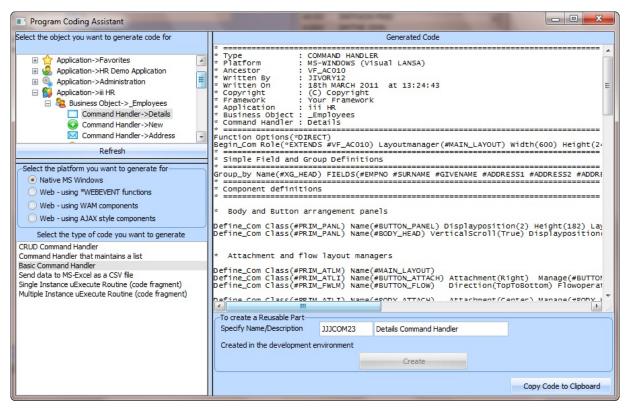
Program Coding Assistant			Street Street		
Select the object you want to generate code for	Specify fields that	you want ont	Basic Command Har o the top (header) area and/or b		list or grid) area of your
Application->Favorites     Application->HR Demo Application     Application->HR Demo Application     Application->Administration     Application->Aim HR	command handler p that code assistan	banel. Your ch t structures ye	n the top of your command handl	r and	
Business Object->_Employees     Command Handler->Details	Field Name	Туре	Description	-	Drop Selected
Command Handler->New	1 EMPNO	ALPHA	Employee Number	н	Drop All
Command Handler->Address	2 SURNAME	ALPHA	Employee Surname		
Refresh	3 GIVENAME	ALPHA	Employee Given Name(s)	-	
	4 ADDRESS1	ALPHA	Street No and Name	-	
Select the platform you want to generate for	5 ADDRESS2		Suburb or dept	-	
Web - using *WEBEVENT functions Web - using WAM components Web - using AJAX style components	Add fields from this		PSLMST Add Keys	dler-	Add All
Select the type of code you want to generate CRUD Command Handler	Field Name	Туре	Description		Drop Selected
Command Handler that maintains a list Basic Command Handler Send data to MS-Excel as a CSV file Single Instance uExecute Routine (code fragment) Multiple Instance uExecute Routine (code fragment)	Add fields from this	Physical File	Add Keys		Drop All
					<< Back Next >> Cancel

9. Click *Next*>>.

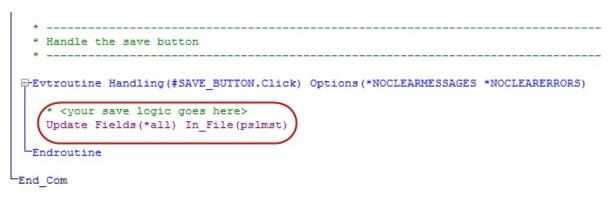
10.On the next page click the *Generate Code* button.

The next page, *Generated Code*, displays the source code for your command handler. You now need to create the component that will contain the code:

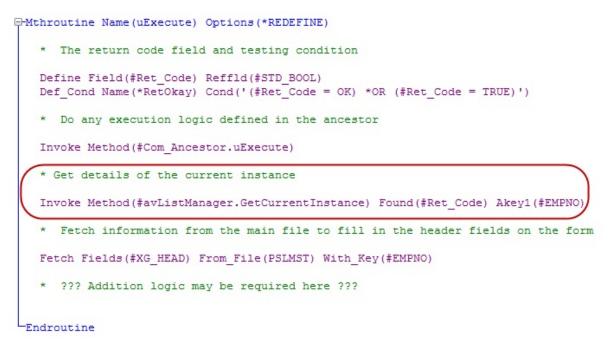
- 11.Specify iiiCOM23 as the name of your component (where iii are your initials) and Details Command Handler as the description.
- 12.Click the *Create* button to create the component.



- 13.After a few moments the *Created in development environment* message will be displayed.
- 14.Switch to the Visual LANSA editor.
- 15.Display the *Source code* of your component.
- 16.Locate the SAVE_BUTTON.Click event and add a statement to save any changes you make to the fields on the Details command handler.



17.Locate the uExecute method. Notice that it calls the #avListManager.GetCurrentInstance method to get the key value of the currently selected item in the instance list and then uses this key value to fetch the details.



18.Compile your component.

#### Step 2. Snap in your Command Handler

Now that you have compiled your new reusable component (that is, your *Command Handler*) and are ready to test it you need to snap it into the Framework.

- 1. Switch to the Framework.
- 2. Select the iii HR application and display the properties of the Employees object by double-clicking it.
- 3. On the resulting properties dialog, click the *Commands Enabled* tab.
- 4. Select the Details command.
- 5. Select the *Component* option in the *Windows* group box.

Click the *Search* button and use the *Find* dialog to locate your command handler. Select the component and click *OK* to select it and return the *Identifier* to the framework. Components must be snapped into the framework using *Identifier* not *Long Name*.

Se Business Object Propert	tiesEmployees	
To enable and disable commar lists Not Enabled	Enabled	Details (DETAILS) Choose Command Type
About Framework	Address Details New	Business Object Command     Instance Command       Sequence:     1
All Details All Entries E Amount Approve Assess Assistant Exampl Assistant Exampl Assistant Exampl Assistant Exampl Attach Attach Attach Attach Basic details	🦻 Skilis	Command Options       Own Window Size         Stay Active       Default          Default Command       Yes          Image: Allow on Web       Windows         Image: Allow in Windows       Image: Optional Arguments         Image: Allow on Popup Menus       Optional Arguments         Image: Show on Instance List Tool Bar       Alpha Argument 1:         Image: Hide All Other Command Tabs       Numeric Argument 1:         Image: Restricted Access       Numeric Argument 2:         Image: Execute as Hidden Command       Numeric Argument 2:
Bookings Calculator Calculator Calendar Calendar Cancel Card Card Category Category Category Category		Command Handler Vindows Component IIICOM23 Mock Up - RAD-PADRADPAD1D687C477BD04D4C9A806489F291F647.HTM Web Browser WEEVENFKLidden EuropeanFEIDPC

- 6. Close Employee properties dialog.
  - a. Select the iii HR application and the Employees business object.
  - b. Click the Search button to populate the instance list.

- c. Select one item in the instance list to display the instance commands.
- 7. Your command handler for Details is now snapped into the Framework and is usable.

<b>19</b>		_Employees		- 🗆 🗙
<u>File Edit View H</u> elp <u>W</u> indow	s			
New Reports Web Site Abo	Address Resources Org	anization Bookings Charges Spool F	iles New Window	Quick Find
_Employees		x		
Favorites     Generation	By Name Dy Start D		🖂 🗆 🕥 👂	X
iii HR Application     Section     Section	Employee Surname S	Search	Number Full Name A0193 Smithson	
<ul> <li>Statistical Reporting</li> <li>Programming Technique:</li> </ul>	✓ Clear List		A0907         Simpson           A1002         Smythe J           A1003         Smithe R           A1004         Smithson           A1005         Smiths Pe	lohn obert Ruth
	٢	>	A1005 Smithers	
	🍓 _Employee : Details (Al			×
	Details Address	> Skills		
	Employee Number	A0907		Start date () Save
	Employee Surname	Simpson		Termination
	Employee Given Name(s)	Anne		Department
	Street No and Name	33 Anne street		Section Code
	Suburb or Town	Annevile		Employee Sa
	State and Country	NSW		Start Date (I
	Post / Zip Code	2145		Termination
	Home Phone Number	090909		Monthly Sala
	Business Phone Number	090909		
< >				
🖃 💷 📴 🏠 🏹 1		Messages Ready	Local ENG	JIVORY13 5/12/13 16:44 🥥

8. Try making a change to the details of an employee and saving it.

#### Summary

# **Tips & Techniques**

- To understand how the command handler interacts with the instance list, read Filter and Command Handler Programming.
- The source code for the command handlers used in the demonstration application can be found in the repository in components named DF_*.

# What I Should Know

- What you need to do to create your own command handlers.
- How you snap them in the Framework.
- Filters and Command Handlers are just Reusable Parts which you can customize. However, you can see that up to this point you can get a functional application simply using the *Program Coding Assistant* without very much coding
- To use the Framework you need to understand VL. However, the level of detail that you must understand is greatly reduced. Creating your own framework to deliver this style of application requires detailed OO knowledge and can take a long time to produce. The VLF allows you to rapidly prototype and deploy an application with no OO knowledge required.

# LVF080WIN - Add Instance List Columns in Windows Applications

# Objective

• Learn how to add columns to an Instance List in a Windows application.

<b>%</b>	_Employees _ 🗆 🗙
<u>File Edit View H</u> elp <u>W</u> indows	(Framework) (Administration)
New Reports Web Site About	Address Resources Organization Bookings Charges Spool Files New Window Queues
_Employees	
Favorites     General HR Demo Application     HR Demo Application	Search     N     Full Name     Depar     Salary       ADM     ADM     Search     N     Full Name     Depar     Salary       ADM     ADM     Search     AL     Paul Patrick     ADM     2345.82       ADM     Section Code     AL     Paul Patrick     ADM     2345.04       Ion     AL     Paul Patrick     ADM     2345.04       AL     Paul Patrick     ADM     26456.04       AL     Paulos Bradley     ADM     313000.04       Ion     AL     Mode Bradley     ADM       Ion     AL     Mode Bradley     ADM       AL     Noculas Adam     ADM     37000.04       AL     Robinson Mary     ADM     44455.04
ŗ	remework business object

**Note:** In this exercise, you will modify the *By Location* filter. Normally, you would do the same modifications to the *By Name* filter and any other filters for _Employees.

To achieve this objective, you will complete the following:

- Step 1. Add Columns to the Instance List
- Step 2. Change your filter
- Summary

#### **Before You Begin**

• You must have completed all the preceding exercises.

You may wish to review:

• Adding Additional Columns to Instance Lists .

#### Step 1. Add Columns to the Instance List

In this step, you will configure your Employee business object to make the extra columns visible in the instance list.

- 1. Start the Framework as a designer.
- 2. Open the properties of the Employees business object.
- 3. Display the *Instance List Settings* tab.
- 4. Two visual identifiers are already defined. Add two additional columns:

#### **Column Sequence Column Type Column Caption Decimals**

30	ACOLUMN1	Department
----	----------	------------

40 NCOLUMN1 Salary 2

	cons Vis	sual Styles	Filters	Eiter Sett							
Double did				THUR DEC	tings (	Commands Enabled Comm	and Displa	y Custom	Properties	SubTypes	Inst
	ck for defa	ult comman	nd			Save and Restore I	instance Lis	sts			
Allow multi	iple select	ions				Enable Clear List Bu	itton				
Allow Insta	ance List t	to be sent to	o MS-Exc	el		File Prefix to be used	for MS-Exc	el	Spreadshee	et_	
Pop up panel na	ame				Enabl	e Pop Up Panels					
Instance List Too	ol Bar Loca	ation		T	ор						
Instance List Too	ol Bar Tex	tLocation		-	None>			*			
	10				o terre e			-			
		abt or Width									
		ght or Width	1	2	4						
			1	2	4			_	~		
Snan in Tostance	a List Brow		Cap		4	Vidth % (Total 100%	Decimals	Edit	Code	Date/Tin	ne Ou
Snan in Tostance Sequence					4	Vidth % (Total 100%	Decimals	Edit	Code	Date/Tin SYSFMT8	ne Ou
Sequence 10 VISU	Type	icer			4		Decimals		code		ne Ou
Sequence 10 VISU 20 VISU	Type UALID1	Number	Cap		4	10	Decimals	Default	Code	SYSFMT8	ne Ou
Sequence         VIS0           10         VIS0           20         VIS0           30         ACC	Type UALID1 UALID2	Number Full Name	Cap		4	10 60	Decimals	Default Default	: Code	SYSFMT8 SYSFMT8	ne Ou
Sequence         VIS0           10         VIS0           20         VIS0           30         ACC           40         NCC	Type UALID1 UALID2 OLUMN1	Number Full Name Departmer	Cap		4	10 60 10		Default Default Default		SYSFMT8 SYSFMT8 SYSFMT8	ne Ou
Sequence         Instance           10         VISU           20         VISU           30         ACC           40         NCC	Type UALID1 UALID2 OLUMN1 OLUMN1	Number Full Name Departmer	Cap		4	10 60 10		Default Default Default Default	: Cod	SYSFMT8 SYSFMT8 SYSFMT8 SYSFMT8	ne Ou
Sequence         Instance           10         VISU           20         VISU           30         ACC           40         NCC           ACC         ACC	Type UALID1 UALID2 OLUMN1 OLUMN1 OLUMN2	Number Full Name Departmer	Cap		4	10 60 10		Default Default Default Default Default		SYSFMT8 SYSFMT8 SYSFMT8 SYSFMT8 SYSFMT8	ne OL

#### Note:

You may set the initial width of each column as a percentage.

Numeric columns may be given an edit code.

Date columns may have a Date/Time format and UTC setting specified.

#### Step 2. Change your filter

You need to make some changes to your filter to fill the new instance list columns with data.

- 1. Close the Framework
- 2. Open the source of the *By Location* filter (reusable part iiiCOM22) that you created in LVF070WIN Snap in Real Windows Filters.
- 3. Make these changes to the code:
  - a. Change the GROUP_BY command to include the #DEPTMENT and #SALARY fields:

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

b. Locate Select Fields(#XG_Ident) command and change the AddtoList statement to set alpha column 1 and numeric column 1:

* Add instance details to the instance list Invoke #avListManager.AddtoList Visualid1(#EMPNO) AKey1(#EMPNO) Visualid2(#Surname) **AColumn1(#Deptment) NColumn1(#Salary)** 

- 4. Compile the reusable part.
- 5. Start the Framework and test the result.

	x						
🙀 By Name 🥥 By Start Date 📈			O 15				
	_	Number	Full Name	Depart		Salary	
Clear List     Search		A1001	Jones Shirley	ADM	2345.82		
	_	A1012	Paul Patrick	ADM	26456.04		
Department Code		A1013	Pattinson George	ADM	78977.04		
ADM		A1015	Woods Bradley	ADM	313000.04		
Section Code		A1020	Douglas Adam	ADM	121500.04		
		A1021	McCully Lisa	ADM	87000.04		
01		4					

#### Summary

#### What You Should Know

- How to add columns to an instance list.
- How to modify a filter to maintain additional instance list columns.
- You would normally change all filters for the instance list.

# **Appendix A. Personnel Demonstration System**

Personnel System Physical Database Map of Personnel System Sample Data in the Personnel Files

#### **Personnel System**

A business has a very simple Personnel System. The Personnel System allows the company to identify the employees in the company based on the part of the company where the employee works. The Personnel System lists details about the employees and details about their specific skills.

The company has a simple organizational structure. It is divided into departments such as Administration, Audit, Information Services, Legal, Travel, etc. Each of these departments may have one or more sections such as Accounting, Purchasing, Sales, etc. The Department table **(DEPTAB)** stores the list of departments. The Section table **(SECTAB)** is used to store the sections within each department.

The Personnel Master file **(PSLMST)** stores details about each employee. For example, the employee's name, address, and telephone number are stored in this master file. As each employee works in a section of a department, this information is also stored in the Personnel Master file.

Each employee also has a list of skills. For example, an employee might have Cobol, C and C++ programming skills or management and administration skills. A Skills table (SKLTAB) is used to store the skill codes. A Personnel Skills file (PSLSKL) stores the specific skills of each employee.

The Personal Event Log file **(PSLEVENT)** allows significant events and notes to be recorded against an employee. It logically extends the PSLMST file. It is an RDMLX file and therefore will only be available in an RDMLX partition.

The Personnel Time Sheet file **(PSLTIMES)** records employee time sheet details. Details are recorded by week number (1 to 52) within a year for each employee. It is designed mostly for use with L/Client and to show extensive trigger power by performing relatively complex calculations and storing them in the DBMS without the application needing to know what is happening. Note that all the data is created and stored in the DBMS when information is created or updated, which means that L/Client applications have read access to it without needing to use the triggers. It is an RDMLX file and therefore will only be available in an RDMLX partition. It contains examples of a number of RDMLX field types including BLOB.

The Personnel System is a very simple system. It has 7 files as described above. The physical database layout follows.

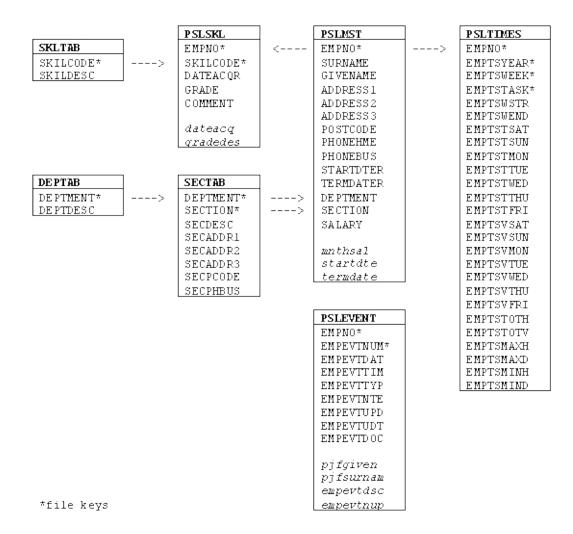
**Historical Note:** This system was created in 1987 as one of the very first LANSA demonstration and training systems. The LANSA repository and

RDML functions created for this original system have been used on a System 38, AS/400, iSeries, System i i5/OS, IBM i Windows, Linux and other platforms. This original system has been left virtually unchanged to show how LANSA has been able to protect your investment in your application systems. (PSLTIMES and PSLEVENT have been added)

#### **Physical Database Map of Personnel System**

(Including Virtual and Predetermined Join Fields)

Sample Data in the Personnel Files



#### Sample Data in the Personnel Files

Following is a list of some of the sample data in the Personnel File which may be contained in the files. As developers can edit these files, the data may not be the same on your system. When the Personnel System is installed the files are populated with the sample data. This can be re-run at any time by executing the Function PSLINI in Process PSLUTL:

	SECTAB: DEPTMENT/SECTION	PSLMST:
DEPTMENT	DEPTMEN1/SECTION	EMPNO
ADM	ADM 01	17 employees
	ADM 02	A1002
		A1005
		A1014
		A8888
	ADM 03	
	ADM 04	
	ADM 05	
AUD	AUD 01	
	AUD 02	
	AUD 03	
FLT	FLT 01	
	FLT 02	
	FLT 03	
INF	INF 01	
	INF 02	
	INF 03	