RAMP-NL Guide

Rapid Application Modernization Process for newlook Guide

- How to Get Started with RAMP
- Licensing Requirements
- Installation and Configuration
- Starting RAMP
- Concepts
- Modernization Issues
- Tutorials
- Scripting
- Screen Wrappers
- Programming Techniques
- Multilingual RAMP Applications
- Troubleshooting
- Frequently Asked Questions
- Movie Index

Please send your comments and suggestions to LANSA Support at: lansasupport@lansa.com.au.

Disclaimer: While every effort has been made to ensure that the information in this material is accurate, in no event shall LANSA be liable for any damages arising from its use. LANSA MAKES NO WARRANTIES, EXPRESSED OR IMPLIED.

Edition Number EPC130100

Edition Date July 16, 2012. © 2012 LANSA

How to Get Started with RAMP

This is the recommended 6 *step plan* for getting started with RAMP:

1. Review the introductory movie What is RAMP? - 9 minutes

2. Comprehend how modernization will change the nature of a 5250 application by reviewing these movies:

5250 Application before using RAMP - 2 minutes

5250 Application after using RAMP - 4 minutes

3. Comprehend how the modernization process is performed by reviewing these movies:

Stage 1: Creating a Modernization Framework - 8 minutes

Stage 2: Snapping the 5250 Application in the Framework - 11 minutes Stage 3: Enrichment and Re-engineering - 5 minutes

4. Install and Configure RAMP and newlook software (see Installation and Configuration).

5. Complete this essential RAMP Tutorial:

Modernizing a Complete Application

6. If you have time, complete this optional RAMP tutorial that deals with a different way to use RAMP for application modernization: Modernizing Application Navigation

Once you have completed these steps you should be well positioned to start to plan and implement your own modernization project using RAMP.

Also see Prerequisite Skills.

Prerequisite Skills

To use RAMP you need to have some basic knowledge of how LANSA and the Visual LANSA Framework are used for application development. You may want to review some of these tutorials (these links are to other guides):

- Framework tutorials
- Visual LANSA tutorials you will need to know how to create, compile and check programs into your System i server using the Visual LANSA editor.

Subject Matter Expertise

To modernize applications with RAMP you need to have access to someone who has an in-depth knowledge of the business application being modernized and the industry in which the application runs. This person should also know what your business is wanting to achieve as a modernized output.

Without access to a subject matter expert you are unlikely to be able to successfully modernize any application using any tool because:

- Nobody would know how the existing functionality is used (as opposed to just understanding how it works, which is different)
- Nobody would be able to envision how the modernized version would be used (as opposed to how it should work).

What is RAMP? - 9 minutes

Play Movie to see what RAMP is.

5250 Application before using RAMP - 2 minutes

Play Movie to see the user view of a 5250 application before it is modernized with RAMP.

5250 Application after using RAMP - 4 minutes

Play Movie to see the same 5250 ERP application after it is modernized with RAMP.

Stage 1: Creating a Modernization Framework - 8 minutes

Play Movie to learn how to start modernizing your application by creating a working prototype for it.

Stage 2: Snapping the 5250 Application in the Framework - 11 minutes

Play Movie to learn how to integrate your 5250 application to the Windows navigation framework.

Stage 3: Enrichment and Re-engineering - 5 minutes

Play Movie to learn about the optional stage 3 of RAMP modernization during which you enrich and re-engineer your application. (5 minutes)

What's New

This section outlines new features in EPC868 version of RAMP-NL:

More Information is Now Accessible in In RAMP-TS and RAMP-NL Scripts

The objCommand object now contains details about the optional arguments associated with the current command and the reason that the command is being executed (ie: you can now distinguish between a command execution and a command activation). Refer to the objCommand definition for more details.

To review new features in previous RAMP versions, see:

New features in EPC 831 Version of RAMP

New features in EPC 831 Version of RAMP

This section outlines features that were introduced in EPC 831 version RAMP:

Dynamic Naming of Newlook screens and fields

It is no longer necessary to use Newlook Designer to identify the relevant screens and fields to Newlook prior to using the RAMP tools. Instead you can use Dynamic Naming of Newlook Screens and Fields from within the RAMP tools environment.

Note that newlook licensing features may limit the use of this option in some RAMP environments. Please contact your product vendor for further information.

Developers' Workbench

You can use the new Developers' Workbench to create your RAMP applications.

New Scripting Functions

SETKEYENABLED Function

enables or disables buttons and/or function keys that were enabled or disabled when defining the destination.

SETFOCUS Function- Set the focus to a field on the current screen

GETFOCUS Function - Get the name of the field with focus on the current screen

COPYTOCLIPBOARD Function-

Web applications can override RAMP profile and password

In the Web signon IIP it is now possible to override the user profile and password used to start a RAMP session specified. The shipped version of this is function UF_SYSBR/UFU0001. See the

source code of UFU0001 for more details.

New newlook Server Property Use INI file

Use the Use INI File property to associate a Newlook server to a Newlook ini file.

RAMP application running in a browser now displays scroll bars by default

Up until now, when the size of a Destination screen exceeded the size of its container no scroll bars were shown and part of the Destination screen was hidden. The only option for users to see the hidden part was to resize the command and/or the entire browser container.

Now scroll bars are

Copy a string to the user's clipboard MAKESUBFILEINTOSTRING Function - Return a DataGrid as a string

SET_UNKNOWN_LOCKING Function - Override Session lock property

FATAL_MESSAGE_TYPE Function – Stop the Framework shutting down when a fatal navigation error occurs

New deployment options

You can now specify the Update File and Codebase values in the server definition instead of modifying the VF_SY120.js file.

Alternatively you can specify them as as URL parameters when starting your application:

+NLCODEBASE= +NLUPDATEFILE=

Optional command arguments accessible in RAMP scripts

It is now possible to reference the two alphanumeric and two numeric optional command arguments in RAMP scripts. Use the context menu (right click) Current Command to paste the values into your script. For example:

objCommand.uAlphaArg1 objCommand.uNumArg2 automatically shown.

newlook version

This version of RAMP requires newlook Version 8.0.5.14307 (or later)

Dynamic Naming of Newlook Screens and Fields

Previously you were required to use newlook Designer to identify the screens and the fields in your application before you could start modernizing it using RAMP. Now the new Dynamic Naming feature allows you to set the name of newlook screens and fields in the RAMP Tools window itself:

Dynamic Naming				
RAMP Screen (Name)		uPersonnelSysMain		
Input Controls Output Contro		s		
At Row / Column	At Row / Column Preceding Label / Content Current (Name) Assigned			
5/2	Enter nur	nber of function required or place ci	utxtMenuOption	
5/6	Full Empl	oyee Listing		
6/6	Enrol a N	ew Employee		
7/6	Browse/N	Maintain Employee and Skill Files		
876		e Business Phone Number List		
976	-	Salary Commitment Reports		
10/6		àeneral Employee Search		
11/6	· · · · · · · · · · · · · · · · · · ·	ne Number Search		
12/6	<u> </u>	ction/Employee Window Enquiry		
13/6		el Table Maintenance Menu		
14/5	Display process or function HELP text			
15/5	Exit from system			
	Г			
Last Saved 25th JULY 2008 at 14:41:18				
User profile		DCXUSER		
When showing a different screen the tracking area should add a new rectangle. If not, it is most likely the screen has to be identified using Newlook Designer.				
				Save

This means that if you use dynamic naming, it is no longer necessary to start newlook outside RAMP, provided that the default screen IDs created by newlook uniquely identify the screens in your application.

Before you start naming the screens, you may want to use the Identify function in newlook tools to ensure that your screens have unique Screen IDs. If they do not, you will not be able to give them unique names.

Note that newlook licensing features may limit the use of this option in some RAMP environments. Please contact your product vendor for further

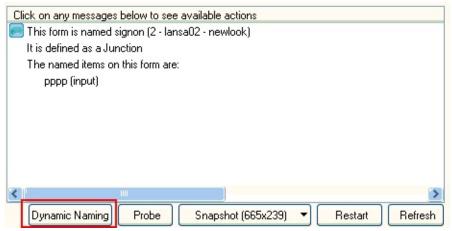
information.

Using Dynamic Naming Dynamic Naming Dialog Details Frequently Asked Questions Backing Up Screen Definitions New IIPs for Windows

Using Dynamic Naming

Before choreographing the navigation in your application, you need to name all the screens in your application.

Using RAMP tools, start newlook and connect to your newlook session. When a screen is displayed in the newlook window, the *Dynamic Naming* button is shown under the RAMP messages area:



When you click the *Dynamic Naming* button, the Dynamic Naming dialog is displayed. If the current screen has not been named previously, the dialog looks like this:

At Row / Column	put Controls	Current (Name) Assigned		
	Preceding Label / Content	Current (Name) Assigned		
5/2	Enter number of function required or place ci			
5/6	Full Employee Listing			
6/6	Enrol a New Employee			
7/6	Browse/Maintain Employee and Skill Files			
8/6	Employee Business Phone Number List			
9/6	Produce Salary Commitment Reports			
10/6		Perform General Employee Search		
11/6	Telephone Number Search			
12/6		Dept/Section/Employee Window Enquiry		
13/6	Personnel Table Maintenance Menu			
14/5	Display process or function HELP text			
15/5	Exit from system			

Type in the RAMP Screen (Name). Define any Input or Output fields on the screen.

Save the screen details by clicking on the Save button.

When you have saved a screen with a name, the Dynamic Naming dialog shows when the screen was named and the user that named it:

🔲 Dynamic Nami	ing			
RAMP Screen (Name)	e) uPersonnelSysMain			
Input Controls Output Controls				
At Row / Column				
5/2	Enter number of function required or place ci_utxtMenuOption			
5/6	Full Employee Listing			
6/6	Enrol a New Employee			
7/6	Browse/Maintain Employee and Skill Files			
8/6	Employee Business Phone Number List			
976	Produce Salary Commitment Reports			
10/6	Perform General Employee Search			
11/6	Telephone Number Search			
12/6	Dept/Section/Employee Window Enquiry			
13/6	Personnel Table Maintenance Menu			
14/5	Display process or function HELP text			
1575	Exit from system			
Last Saved	25th JULY 2008 at 14:41:18			
User profile	DCXUSER			
When showing a different screen the tracking area should add a new rectangle. If not, it is most likely the screen has to be identified using Newlook Designer.				
		Save		

You should name all the screens in your application before you start tracking the navigation.

Dynamic Naming Dialog Details

RAMP Screen Name	This is the programmatic name that RAMP uses to uniquely identify this screen as a destination, a junction or a special screen. It is stored in the (Name) property of the screen inside newlook. This is why it is displayed in brackets. Usually every newlook Host Screen Name has a unique RAMP (Name) associated with it. Sometimes, in situations where multiple newlook Host Screens are very similar, they are all assigned the same RAMP Screen (Name) to reduce the amount of scripting required. Refer to the guide for more details about this special programming technique.
Input Controls	 The input controls tab sheet shows the list with all the input controls found in the Newlook screen currently showing. Note that input controls are not only input fields. Use the <i>Current (Name) Assigned</i> column to modify or set a name for a control. Notice two things when you focus on a cell in this column: The background of the control in the Newlook screen corresponding to the cell being focused on becomes pink to provide visual feedback as to which field will be named.
	• The cell's content is pre-filled with a default prefix. It allows you to standardise the field names should your application use a certain naming convention. You can modify the default behaviour in the Windows IIP (UF_SYSTM unless you have created your own). See New IIPs for Windows.
Output Controls	The output controls tab sheet shows the list of output controls found in the current Newlook screen. You normally do not need to name output

	fields.
Last Saved	Shows the date and time when this screen was last saved using Dynamic Naming. If this was the first time Dynamic Naming was used for this screen you would see a message indicating so.
User Profile	Shows the User profile that last saved this screen using Dynamic Naming.

Frequently Asked Questions

When should I use newlook Designer to manually name screens instead of using Dynamic Naming?

For troubleshooting purposeses it is important that you understand how the screens are named in newlook. However, you would normally identify your screens using RAMP dynamic naming because it is quicker and easier.

I navigate to a screen which I have not named yet, but RAMP shows it with a name of another screen I have already named?

Newlook has assigned this screen the same Screen Id as another screen. You need to start newlook and use the Identify function to change the Screen Ids so that they are unique.

Backing Up Screen Definitions

You may want to keep backup copies of your screen definitions as a precaution before you use Dynamic Naming.

To do this, select the option *Keep newlook sid file versions (RAMP only)* in the Framework Details tab in Framework Properties:

Keep Newlook SID file versions (RAMP only)

When this option is selected, you will be prompted to save the definitions when you start newlook in RAMP tools:

Keep Newlook SID file versions				
The files in the list below were found in Newlook's installation path C:\Program Files\looksoftware 8.0				
Select the files to be versioned and click OK.				
Name of file version to be saved From Newlook SID file				
DEFAULT_20070925_145925.sid.sav	Default.sid			
NEWLOOK_20070925_145925.sid.sav	newlook.sid			
USER_20070925_145925.sid.sav	User.sid			
	OK Cancel			

New IIPs for Windows

New IIPs have been made available to support Dynamic Naming:

AvMakeFormName	This method is called when a form has yet to be named using Dynamic Naming.
AvMakeControlName	This method is called each time a cell in the Input/Output control grid in Dynamic Naming interface receives the focus. It allows you to standardise the screen's field names should your application use a certain naming convention.
AvValidateFormName	Validate the name given to a Newlook screen.
AvValidateControlName	Validate the name given to a Newlook screen's control.

Example

You can modify the default behaviour of the Dynamic Naming dialog in the Windows IIP (by default UF_SYSTM).

This example sets the prefix for all newlook controls XXX:

•••

* This method is called by the dynamic naming tool to set a default value to give to an unnamed Newlook form.

* Here you can specify a value based on some naming standard to give to all Newlook forms. The default value returned by the ancestor

* is blank.

* To specify your own, comment out or delete the Invoke

Method(#COm_Ancestor.avMakeFormName) Formprefix(#FormPrefix) command

* and insert your own logic. Return the value in the output parameter #FormPrefix.

Mthroutine Name(avMakeFormName) Options(*REDEFINE) * Define_Map For(*output) Class(#vf_eltxtm) Name(#FormPrefix) Desc('Prefix to apply to unnamed forms ')

Invoke Method(#Com_Ancestor.avMakeFormName) Formprefix(#FormPrefix)

Endroutine

* This method is called by the dynamic naming tool when the focus is set into a cell in the grid that shows

* all the *controls in the Newlook form that is showing. Here you can specify a value perhaps based on some

* naming standard to give to all defined Newlook controls. The default value returned by the ancestor blank.

* To specify your own, comment out or delete the Invoke

Method(#COm_Ancestor.avMakeControlName) Controlprefix(#ControlPrefix) command

* and insert your own logic. Return the value in the output parameter #ControlPrefix.

Mthroutine Name(avMakeControlName) Options(*REDEFINE) * Define_Map For(*output) Class(#vf_elctln) Name(#ControlPrefix) Desc('Prefix to apply to unnamed controls ')

* Invoke Method(#COm_Ancestor.avMakeControlName) Controlprefix(#ControlPrefix)

set com(#Controlprefix) value(xxx)

Endroutine

* Use this method to validate the name given to a Form using the Dynamic Naming tool. The default behaviour is to return OK

* except when the form name is equal to the value set in method avMakeFormName:

- * If Cond('#FormName = *blanks')
- * Set Com(#ReturnCode) Value(ER)
- * Set Com(#ErrorMessage) Value(*MTXTVF_UM701_014)
- * Else
- * Set Com(#ReturnCode) Value(OK)
- * Endif

Mthroutine Name(avValidateFormName) Options(*REDEFINE)

* Define_Map For(*input) Class(#vf_eltxtm) Name(#FormName) Desc('Form name to be validated')

* Define_Map For(*output) Class(#vf_elretc) Name(#ReturnCode) Desc('OK or ER')

* Define_Map For(*output) Class(#vf_elmsg) Name(#ErrorMessage) Desc('Returned message in case of error')

Invoke Method(#COm_Ancestor.avValidateFormName) Formname(#FormName) Returncode(#ReturnCode) Errormessage(#ErrorMessage) Endroutine

* Use this method to validate the name given to a Newlook control using the Dynamic Naming tool. The default behaviour is

* to return OK except when the control name is equal to the value set in method avMakeControlName:

* If Cond('#ControlName = *blanks')

- * Set Com(#ReturnCode) Value(ER)
- * Set Com(#ErrorMessage) Value(*MTXTVF_UM701_013)
- * Else
- * Set Com(#ReturnCode) Value(OK)
- * Endif

Mthroutine Name(avValidateControlNam) Options(*REDEFINE)

* Define_Map For(*input) Class(#vf_elctln) Name(#ControlName)

Desc('Control name to be validated')

* Define_Map For(*output) Class(#vf_elretc) Name(#ReturnCode) Desc('OK or ER')

* Define_Map For(*output) Class(#vf_elmsg) Name(#ErrorMessage) Desc('Returned message in case of error')

Invoke Method(#COm_Ancestor.avValidateControlNam) Controlname(#ControlName) Returncode(#ReturnCode) Errormessage(#ErrorMessage)

Endroutine

•••

New features in EPC 826 Version of RAMP

This section outlines new features in EPC 826 version RAMP:

New Subfile Accessor properties for faster scripting

For subfiles that use markers such as "+", "More..", "Bottom":

- The new *EndofFileMarker* property indicates the subfile end marker. The default is "Bottom".
- Scrolling stops when either no marker is found, or when the marker exactly matches EndofFileMarker.

For subfiles that issue a message when attempting to scroll past the end:

 If the subfile has no marker at all, set the new UseMarker property to False and EndofFileMarker to text that can be found somewhere in the message that appears when attempting to scroll past end. For example EndofFileMarker
 = "made to scroll past end"

See the Properties table in SUBFILE_ACCESSOR Object

newlook Version

This version of RAMP requires newlook Version 8.0.2.11391 dated July 18 2007 (or later).

Add your own options to the scripting pop-up menu by creating an xml file

See Adding Your Own Options to the Scripting Pop-Up Menu.

Set Command Handler Caption from RAMP Scripts

The captions for command handlers can now be changed from RAMP scripts using the new SET_HANDLER_CAPTION Function.

Turn off recursion checking for RAMP scripts

You can use the new global property *GLOBAL_flagRecursionCheck* to turn off recursion checking. See Switching Off Recursion Checking.

Easier liteclient license configuration

Newlook liteclient licenses can now be used without having to manually modify any HTML file.

New features in EPC 804 Version of RAMP

This section outlines new features in EPC 804 version RAMP:

Wrappers for 5250 screens

Using RAMP Screen Wrappers you can present a good looking, easy to use, high GUI veneer over 5250 screens without having to analyze and rewrite all the business logic imbedded inside them.

newlook function key bar

The newlook function key bar may now be optionally displayed on RAMP screens using the OVERRIDE_BUTTONS_UNDEFINED_SCREENS Function.

This is most commonly done in pop-up windows.

Virtual Clipboard Access

Information placed onto the virtual clipboard by VLF filters and command handlers can now be read and updated from RAMP scripts using the AVSAVEVALUE Function and the

AVRESTOREAVALUE and AVRESTORENVALUE Function. This significantly improves the ability for RAMP scripts and filters and command handlers to exchange information.

Available in VLF web or windows based applications.

User-defined lock message

The message that appears when a user attempts a RAMP navigation from an unknown screen has been improved.

The message presented is now different for designers and end-users. Additionally the message text may now be changed from a RAMP script to

newlook Version 8

This version of RAMP re Version 8.0.1.10669 date 2007 (or later).

Set multilingual functic

The captions show on Ra now be changed to be mu the

OVERRIDE_KEY_CAF Function and OVERRIDE_KEY_CAF Function.

Scripts

Commonly used script lc placed into a common Ja User-defined script funct better reuse.

Script names may now b Creating a Script Namin sometimes necessary wh developers work on the s Also see xxxxxx is an c should be deleted.

Handling undefined sci

Often unknown screens a display by users, causing screen lock message to b when they attempt to nav somewhere else. Now yc Windows RAMP applica exactly what any site requires using the SET_LOCK_MESSAGE Function.

Deployment

Visual LANSA Framework and RAMP check lists for deployment are available. The detailed check lists guide the user in planning the deployment, packaging the material and installing the package.

Handling Pop-ups

A new section Handling Pop-Ups describes how to handle pop-up windows in RAMP. The new FORCE_POPUP_REFRESH Function displays hidden pop-ups. 'guess' what it might do make an unknown screer the ADD_UNKNOWN_ Function, so that the user request can be handled c

Change Date/Time/Use

The RAMP tool now dis changed date-time-user f and scripts to make scrip management easier.

List of objects in the sess sorted by their date and t

Advanced Prompting

Information is provided (the additional informatio passed to or retrieved frc form using the HANDLI Function.

New features in EPC 793 Version of RAMP

This section outlines new features in EPC 793 version RAMP:

Performance

The start-up times for Framework applications (including RAMP) have been substantially improved in this version. For more information see What's New in the Framework Guide.

Instance Lists

RAMP-specific instance list information is now provided. See Advanced Instance List Processing.

To review new features in previous Framework versions, see: New features in EPC 785 Version of RAMP

New features in EPC 785 Version of RAMP

RAMP in the Browser

RAMP applications can now be executed in the browser version of the Visual LANSA Framework. You can use both Webevent and WAM filters in your application.

See Starting the Framework on the Web

Merge Tool

Framework and RAMP applications can now be merged together When Many Developers Work on the Same Application.

New Choreographing Features

You can use the Snapshot button while choreographing to capture the 5250 screens as a bitmaps.

You can use the screen images to give your prototype a realistic look, as a memory jogger or for documentation purposes during POC exercises.

Also, a new Refresh button is available to force the RAMP choreographer to re-examine the current 5250 screen. This is useful in situations involving 5250 flash screens

Internet Explorer (IE) 7 is now supported as a browser in VLF end-user applications

Note that at this date IE7 is still a Microsoft beta product and it is not yet supported by the Visual LANSA IDE for developers.

Scripting

New functions: GET_MENU_OPTION_NUMBER Function STRIP_LEADING_NUMBERS Function ADD_STRING Function

Subfile Handling Enhancements

Subfile access from scripts has been improved. A new SUBFILE_ACCESSOR object is provided and simpler scripting is now possible.

See Subfiles/Browselists for

STRING Function

The automated generation of scripts has been improved, with better explanations in cases when a script cannot be generated automatically

Also see Javascript Essentials documentation and the Movie Index for new scripting movies.

Run-Time RAMP Performance Improvements

Business Object instance list processing is faster in all Visual LANSA Framework Windows applications.

RAMP application start-up performance has been improved, particularly when executing in enduser mode.

The 5250 Home key is now handled in RAMP applications

movies on subfile handling. A new Probe button is available to probe the current 5250 form layout, especially subfile layouts. It presents information about the fields and subfile rows and columns that makes scripting easier and simpler.

Changes To Trace and Messaging

Application Tracing and message handling have been improved, including the ability to save trace details to a file.

Buttons to turn application level tracing off or on are now presented directly on RAMP screens when working as a developer as continual reminder that this is the primary script debugging aid.

User Interface Improvements

The newlook 5250 session height, width, location and masking areas can now be set for individual 5250 forms to override the session level defaults in Destination Screen Details.

Licensing Requirements

Developer Quick Check List

If you are RAMP developer then use this quick check list:

Checked	Type of License	What does it allow you to do	How to check it
	VL-IDE dongle or Softkey license	To develop Visual LANSA applications	Can you create, compile and execute Visual LANSA (VL) applications and check them into your System i server?
	RAMP choreographer license	To choreograph RAMP screens	Use LANSA REQUEST(LICENSE) on your System i server and look for a valid "AXX" or "Ann" license.
	newlook Professional Edition (PE) or Centric license	To identify and enhance screens using the newlook Designer.	Use WRKLICINF PRDID(0NWLOOK) on your System i server.
	newlook smartclient or liteclient license	To execute 5250 applications inside newlook.	WRKLICINF PRDID(0NWLOOK) on your System i server.
	LANSA Super Server license	To access remote data and programs on your System i via non- 5250 interfaces.	Use LANSA REQUEST(LICENSE) on your System i server and look for a valid "LXX" or "Xnn" license.

To obtain any type of license contact your LANSA product vendor. If you need to understand the detailed licensing requirements for both developers and end users, refer to Complete Licensing Details. More About Newlook Licensing describes newlook licensing.

Complete Licensing Details

License Type	RAMP license	Visual LANSA Development license	newlook Professional Edition (PE) or Centric license	Newlook smartclient or liteclient license	LAN Supa Serv licen
License Purpose	To choreograph RAMP screens	To develop Visual LANSA applications	To identify and enhance screens in newlook	To execute RAMP applications	To a remc data prog on y Syst via n 525C inter
Required by RAMP application developer	V	v - See Note 1	V	V	v - S Note
Required by RAMP application end-user				V	v - S Note

Notes:

1: Visual LANSA Development licenses are dongles or soft keys that are bound to a specific developer workstation. All other licenses are slot-based (licensed per server by number of concurrent developers).

2: Visual LANSA Development licenses may include a Super-Server license. Entitlements may vary.

3: RAMP applications may be designed and developed in a restricted manner so that they do not require LANSA Super Server licenses.

Visual LANSA Version

Visual LANSA version 12.0 with EPC 859 and patch EPC859HF-101005 applied, or later.

newlook Version

You need to use newlook Version 8.0.5.14769 (or later).

About the RAMP License

The RAMP license is only required to use RAMP tools during application development. It may be an AXX (unlimited) or an Ann (limited to nn concurrent developers) license.

Install it on the System i server that you use for application development.

If you do not have a license to use the RAMP tools you need to:

- Obtain a RAMP license from LANSA.
- Input the license code on your System i server using the LANSA REQUEST(LICENSE) command followed by the *Insert a Permanent License Code* menu option. Key in the License Type (AXX or Ann) and the License Code assigned to you by LANSA and press Enter. This method of license installation is the same as for all other licensed LANSA products.
- When you start the Visual LANSA Framework and need to use the RAMP tools, use should use super-server mode and connect to your System i server so that your license can be checked, so a Super Server license is also required.

About the newlook Licenses

- Developers designing and enhancing newlook screens also require use of a newlook execution license (smartclient or liteclient).
- newlook liteclient licenses restrict the range of newlook execution time facilities available.

More About Newlook Licensing

VLF.RAMP starts newlook in two different ways:

- In RAMP tools to support developer choreographing
- On command handler tabs as part of executing an VLF application.

How a LiteClient License is Determined

When newlook is started, this VLF server definition setting is checked:

-newlook- Use 'liteclient' license		
Use INI file	*DEFAULT	
Load Path	*DEFAULT	

In RAMP tools à If any server definition has 'Use liteclient license' ticked, newlook is requested to start up using a liteclient license.

On command handler tabs à If the *connected server* has 'Use liteclient license' ticked, newlook is requested to start up using a liteclient license.

Other Types of Newlook Licenses

If a liteclient license is not to be used, then VLF_RAMP starts newlook without specifying a license type. *This means that newlook will use its own logic to determine what type of license should be used*. It will do this based on how your newlook environment is configured, which is usually determined by you when you installed or upgraded newlook.

Checking the Type of Newlook Licence(s) you have

To verify the type of license used by a client PC you have to check the value of *Edition* in the following registry key:

```
HKEY_CURRENT_USER\Software\looksoftware\newlook\8.0
```

If set to **Host** the client will request a license from the host.

If set to blank the client will has been set to use a **Local** license. A Local licence will usually override all other settings and requests.

To check your **Host** licences use the WRKLICINF PRDID(0NWLOOK) command.

All different license types are listed for the appropriate version but that doesn't mean you the host has a real license. For example, you might see something like this:

Product Term Feature Description			
0NWLOOK	V8R0	5001	smartclient
0NWLOOK	V8R0	5002	newlook
0NWLOOK	V8R0	5003	centric
0NWLOOK	V8R0	5004	soarchitect
0NWLOOK	V8R0	5005	liteclient
0NWLOOK	V8R0	5010	lookdirect

To find out which of the license types the machine is licensed to, press F11. You will then see something like this:

Product Term	Featu	re Lim	it	Count
0NWLOOK V	8R0	5001	0	.00
0NWLOOK V	8R0	5002	3	1.00
0NWLOOK V	8R0	5003	0	.00
0NWLOOK V	8R0	5004	0	.00
0NWLOOK V	8R0	5005	8	.00

The Limit column tells you total number of license seats and the Count column how many seats are currently in use.

Forcing Newlook to start using a specific licence type

Provided that you are not using a local license, you can do this by modifying VLF JavaScript files VF_UM703.JS (used by RAMP Tools) and/or VF_SY120.JS (used to execute newlook on command handler tabs). In both these files you will find a section of code like this:

```
if (flagLiteclient)
{
```

```
var strHTML = "<object id='__objNewLookAX'
onreadystatechange='VF_SY121_KICK_OFF()' style='height:100%;'
width='100%' classid='CLSID:CFFE5E18-79B9-431C-8CE2-
AE55A16E7C09'><param name='options' value='-vs -vr -q -i" + sUseIni + "'>
<param name='TimeOut' value='0'><param name='HideToolbars' value='-1'>
<param name='HideConnectionDialog' value='1'><param name='license'</pre>
value='liteclient'><h1 id='NL_Failed'>Newlook has failed to Initialize.</h1>
</object>";
 }
 else
 {
   var strHTML = "<object id='__objNewLookAX'</pre>
onreadystatechange='VF_SY121_KICK_OFF()' style='height:100%;'
width='100%' classid='CLSID:CFFE5E18-79B9-431C-8CE2-
AE55A16E7C09'><param name='options' value='-vs -vr -q -i" + sUseIni + "'>
<param name='TimeOut' value='0'><param name='HideToolbars' value='-1'>
<param name='HideConnectionDialog' value='1'><h1 id='NL_Failed'>Newlook
```

```
}
```

has failed to Initialize.</h1></object>";

The first section is used to specify a newlook liteclient licence when starting. You need to add a <param name='license' value='smartclient'> string into the code section to indicate what type of license you want to force newlook to use. The second section assembles the HTML tag used to start newlook with a default license. *This is the code section you need to modify*

Installation and Configuration

Installation Configuration Starting the Framework on the Web When Many Developers Work on the Same Application

Installation

Install RAMP Install newlook

Install RAMP

See the Framework Guide for detailed instructions for installing the Framework. You need to:

- Install the Framework software on the System i Server. Note that you only need to perform this step and none of the other steps described in the Framework Guide because you are not installing LANSA for the Web.
- Install and Configure the Framework on Visual LANSA Workstations.

Install newlook

You need to use newlook Version 8.0.5.14769 (or later).

Locate it on www.looksoftware.com.

We recommend that if you have an earlier newlook 8 version installed on the machine, you uninstall it and reboot before installing the new version.

During the installation choose the *Typical* setup.

Next, if you have a newlook license, select the *request host license* option.

Configuration

Verify newlook Installation Verify Internet Explorer Security Settings Configure newlook Configure RAMP

Verify newlook Installation

Start newlook in the *looksoftware suite* program folder. Select option *newlook* 8.0.

Then select the *Connect* option in the *Session* menu.

Connect	
Select the computer	e connection you want to use to access the host
Description	Туре
Add	Remove Properties Wizard On
Show this dia	log at startup

In the Connect dialog click on the Wizard On toggle button to turn it off. Then use the Add button to add the server that has the application you are going to modernize to newlook.



Click Next. Then specify a name for the connection:

Connection Wizard	
	Each connection is identified by a short description and an optional icon.
	Enter a short description for the connection
	Connection 1
	Enter or locate the icon file for the connection
	C:\Icons\All icons\082-server_1.ico
	< <u>B</u> ack Next > Cancel

Click next. Then select the *connection type*:

Connection Wizard	
	you can define an standard connection to the host, or you can treat a group of connections as if they were a single connection.
	Select the connection type Single
	< <u>B</u> ack Next > Cancel

Click Next. Then specify the *internet address*. Do not select *Secure connection*.

Connection Wizard		×
	Specify the internet address to use for the connection.	
	Enter the internet address	
	61.88.200.002]
	Secure connection	
	The internet address can be found by entering WRKTCPSTS OPTION(*IFC) on the OS/400 command line.	
	< Back Next > Cancel	

Click Next. Accept the default *session type*:

Connection Wizard	
	Select the type of session that you would like to start when you select this connection.
	Select the session <u>type</u> Start an iSeries display session
	< <u>B</u> ack <u>N</u> ext > Cancel

Click Next. You do not have to specify a *device name*. Do not select *Connect using lookdirect*.

Connection Wizard	
	Specify the device name to use for this connection.
	Enter the <u>d</u> isplay device name
	 <u>Auto increment display device name if unavailable</u> <u>Connect using lookdirect</u> The display device name is optional, and if not entered will be assigned by the host.
	< <u>B</u> ack Next > Cancel

Click Next. Select the *locale*:

Connection Wizard		X
	Specify the locale you want to use for this connection.	
	Select the locale English (United States)	•
	< <u>B</u> ack <u>N</u> ext > Cance	1

Click Next. Select the sign-on options for obtaining the newlook license from the System i:

- In most situations we recommend you select the *Specify the username and password* option so that the information required for obtaining the license is stored in the newlook.sid file. In this way later on you will be able to deploy a generic profile with your application and the end-user will not be prompted for the username and password. (If you choose the option *Use the common username and password*, newlook uses the profile stored in the HKEY_CURRENT_USER\Software\looksoftware\newlook\8.0\Sign On registry key on the PC.)
- Make sure that the Automatically signon to display sessions option is NOT selected in any RAMP applications:

Connection Wizard	
	You can signon with the common username and password, or using a specific username and password for this connection.
	Choose from the following options Use the common username and password Specify the username and password Automatically signon to display sessions Host uses case-sensitive passwords
	< <u>B</u> ack Next > Cancel

Click Next. Specify the username and password used to obtain a valid profile so that newlook can retrieve a newlook license from the System i. It is recommended you use a profile that never expires.

Connection Wizard	
	newlook requires a username and password in order to automatically signon to the host and to obtain a newlook license. Enter your <u>u</u> sername xuser 1
	Enter your password
	•••••
	Confirm your password
	•••••
	< <u>B</u> ack <u>N</u> ext > Cancel

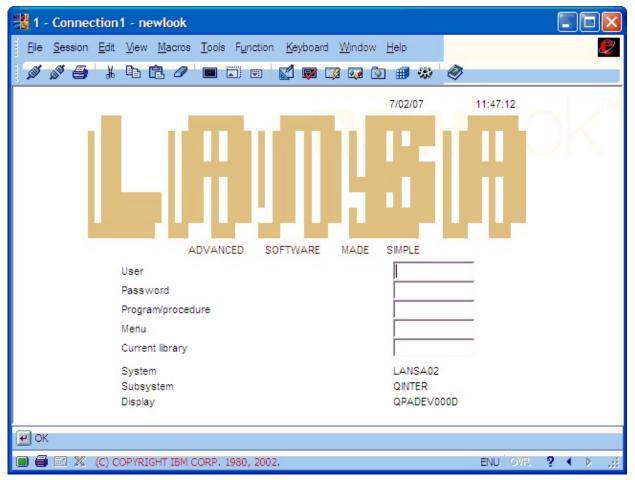
Then click Finish.



When your connection is defined, click on the Connect button. Enter the username and password to sign on to your server:

Signon			X
R	Please enter your use	rname and password.	
	Username Password	 	
	Save password		
۲		OK Cancel	

The newlook version of your server's 5250 sign-on screen is displayed:

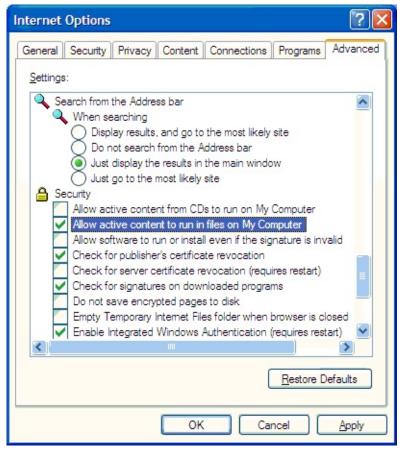


You have now verified newlook has been installed.

Verify Internet Explorer Security Settings

Verify that the internet security setting *Allow active content to run files on my computer* is selected:

- Open Internet Explorer
- Select the *Internet Options* option in the Tools menu
- Display the *Advanced* Tab
- Locate the *Security* group
- Ensure the *Allow active content to run files on my computer* option is selected.



Depending on the version of Internet Explorer you have, this option may not be present. If it is not present, just ignore this step.

Configure newlook

Merge Shipped Macros into newlook Configure newlook for a Windows Look and Feel

Merge Shipped Macros into newlook

Merge VF_XP.nlg Merge VF_MACRO.sid

VF_XP.nlg

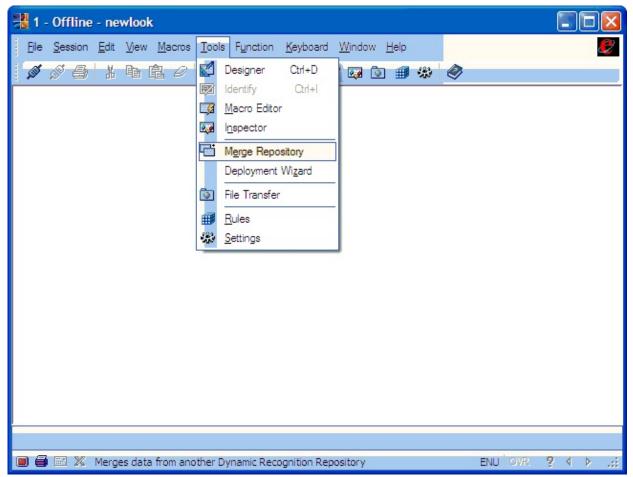
Using Windows Explorer, copy the files of type **VF_XP*.nlg** (VF_XP.nlg, VF_XP_2007BLUE.nlg, VF_XP_2003BLUE.nlg etc):

- **From** the Execute directory of the LANSA partition you will be using (for example C:\Program Files\LANSA\X_WIN95\X_LANSA\x_dem\execute)
- **To** the newlook directory (for example C:\Program Files\looksoftware 8.0)

VF_MACRO.sid

Merge the **VF_MACRO.sid** file into newlook:

- Start looksoftware suite 8.0 then newlook 8.0
- In the Tools menu select Merge Repository



The Browsing for Dynamic Recognition Repository dialog is displayed.

• Use this dialog to locate VF_MACRO.sid in your LANSA partition execute directory (for example C:\Program Files\LANSA\X_WIN95\X_LANSA\x_dem\execute):

Browsing fo	r Dynamic Recognition Repos	itory			? 🛛
Look in: 🧰	execute	V ()		0	.
	D.sid				
File name:	VF_MACRO.sid			Г	Open
-					
Files of type:	Repository Files (*.sid)		*		Cancel

- Click Open.
- Display the Macros tab. Select VF_MACRO from the list of macros. Then click on the Merge button:

Merge Repository	
Forms Macros Connections Filters Subfiles Check Fields Borders Other	
Select the macros to merge. Enter text below to display only macros that contain that text in t description.	heir
List macro descriptions which contain:	1 of 1
Name Author R	evision Ni
VF_Macro	
	>
Select <u>All</u> Sele	ct None
Merge Close	Help

The Framework macros should now have now been merged into newlook.

This is confirmed by a Merge Summary screen like this:

em	il.	Total	\odot	1	0
Forms		0	0	0	0
Macros		1	1	0	0

Configure newlook for a Windows Look and Feel

To optimize the appearance of your RAMP screens, you need to make some changes to newlook options. To do this Start newlook by clicking on the *newlook Designer* button in the RAMP Tools window and then:

- Change the Scheme
- Change the Background
- Ensure newlook uses the Windows Themes
- Suppress newlook Sounds

Change the Scheme

To change the newlook scheme to XP, select the Tools menu and then the Settings option in the newlook window:

📲 4 - Offline - newlook	
<u>File S</u> ession <u>E</u> dit <u>V</u> iew <u>M</u> acros	Tools Function <u>K</u> eyboard <u>W</u> indow
Ø 8 5 1 B B 8	Designer Ctrl+D
	😡 Identify Ctrl+I
	Macro Editor
	🐼 I <u>n</u> spector
	Database Description
	Web Services Description
	Merge Repository
	Deployment Wi <u>z</u> ard
	💿 File Transfer
	🗰 <u>R</u> ules
	Settings

To change the scheme of your RAMP screens, select the Display tab and click on the Settings button in the Appearance area:

newlook Settings - Newlook.ini	
General Display Keyboard Commands Preferences	5
Use these settings to change the way that n forms.	ewlook displays
Appearance You can change the fonts and colors that newlook uses to display graphical forms and emulator screens.	Settings
Background You can change the default background displayed in graphical forms.	Settings
Menus Use these setting to change the way that newlook displays graphical application menus.	Se <u>t</u> tings
OK Cancel Ap	ply Help

In the Graphical tab of the Appearance dialog change the Scheme to VF_XP if your framework application does not use Visual Themes.

If your framework application uses a Visual theme, select the matching VF_XP_* value.

See the Framework property Overall Theme.

pearar Graphical	Emulator	
R	You can change the fonts and colors tha display graphical forms.	t newlook uses to
Schem	e <u>C</u> ategor	у
VF_X	Backgro	ound 🔽
Say	e As <u>D</u> elete	De <u>f</u> ault
-Font-		Sample
Re	Name Tahoma 🔽	
	Size 8 💌 B 🖊 U	
Colors	Host screen color	Display as
٠	Eoreground Green	
	Background Black	
2		

Click OK.

Change the Background

By default newlook uses a background image on the modernized screens which you may prefer remove. To do this, click on the Settings button in the Background area in the Display tab.

The Background dialog is displayed.

Blank out the Picture field to remove the background graphic:

ackground	
You form	a can change the default background displayed in graphical ms.
Picture	
Alignment	Top Right
<u>C</u> olor	Automatic
<u>Q</u> uality	Automatic
Size	Clip
Tile	Texture
Opacity	10
Ref transmut	k color Mask color

Click OK.

Close the newlook Settings dialog by clicking OK.

Ensure newlook uses the Windows Themes

You need to ensure that newlook is set up to use Windows themes and visual styles in order to enable visual effects such as mouseovers. To do this select the Rules option in the Tools menu.

The newlook Rules dialog is displayed:

look Rules - Newlook	.sid		
Check Fields	Borders	Comb	o Fields
General Policies	Categories	Filters	Subfiles
You can specify the create graphical f	ne categories you wa orms.	ant newlook to	use to
Recognition	Menus		
Combo Boxes	MultiLine	e TextBoxes	
Command Combos	Option 8	Buttons	
Date Combos	Prompt Buttons		
Grids	UpDowns		
Hyperlinks	WRLs.		
V Līņes			
Display			
Field Choices	Screen ;	Info	
Field Markers Screen Title			
Output Borders			
ОК	Cancel	Apply	Advanced Help

Display the Categories tab and then click on the Advanced button: The Advanced Categories dialog is displayed:

Advance	ed Categories	×			
• •	Specify the advanced recognition and display categories you want newlook to use to create graphical forms.				
[] Ign	Recognition Ignore unwritten areas of the host screen Create read-only controls				
	y se Windows themes and visual styles nore underline on static fields				
٢	<u>OK</u> <u>Cancel</u>				

Select the Use Windows themes and visual styles option. Then click OK to close the dialog. Close the newlook rules dialog by clicking OK.

Suppress newlook Sounds

By default newlook automatically plays sounds in some situations, for example when a host screen is received. To suppress these sounds:

- Start newlook
- Display the Tools menu and select the Settings option
- Display the Preferences tab in the newlook Settings window and uncheck the *Enable Sounds* option:

newlook Settings - Newlook.ini						
General Display Keyboard Comman	nds Preferences					
You can customize the newlook user interface according to your individual preferences.						
Startup	Windows					
Splash screen	Center main window					
Connection dialog	Center popup windows					
Status bar help	Close popups F3					
Host menus	Confirmations					
 Sounds 	✓ Cancel changes					
Enable sounds	 ✓ Disconnect ✓ Exit 					
Advanced						
OK Cancel Apply Help						

• Click OK

Configure RAMP

Perform these steps to define a newlook server in the Framework: Specify Server Details Set up Super-Server Session Optionally Set up Framework Users and Security Optionally Configure newlook User Profile and Password in the Framework

Specify Server Details

- Start the Framework.
- In the Administration menu of the Framework select the Servers option.
- In the Server Details tab, select LANSA for System i + newlook as the Server Type. (If your System i and newlook servers have different IP addresses even though they might be the same physical server, choose the newlook Only option.)
- Enter the name of the newlook connection as defined in the newlook Connection Properties panel in the Server Name property. You can leave the name blank in which case the IP address and Port Number will be used.
- If you are using a newlook liteclient license, select the Use 'liteclient' license check box.
- If you leave also the IP Address and Port Number blank, the newlook connection panel will be displayed when the Framework is trying to establish a connection.

Server Details Identification Icons			
Server Type:	LANSA for System i + newlook		DBCS Capable
Server Name:	LANSA02		Commitment Control
Partition	DEM		
Client-Server Translation Table:	*JOB		Divert Locks
Server-Client Translation Table:	*JOB		Partition is enabled for RDMLX
Selection Block Size:	50		
Selection Limit:	10,000		
-Windows or Unix Databases			
Database Name:			
Database User:			
Database Password:			
Database Type:	ASA (for all Intel platforms)		
Server Overrides:			
IP Address	999.99.999.999		
Port Number	23		
Use 'litedient' license			
Load Path	*DEFAULT		
L			
		LANSA	Comms. Admin Close

Set up Super-Server Session

When using RAMP you need a super-server session to sign on to the System i server. To specify the sign-on option:

- Start the Framework.
- Display the Framework menu and select the Properties option.
- In the Framework Properties, select the User Administration Settings tab.
- Select the Users Sign on to a Remote Server to Use the Framework option in Sign on Settings.
- Close the dialog and save the Framework.

Help About	Web Details	Developer Preferences - is	Series Server	Develop	er Preferences - Web Server 2	User Administration Settings
Authority	/ Settings					
_	-	ers and Authority				
Stor	e Users in DBM	IS Tables VFPPF06/7				
Store Users in XML File Named			Vf_Sy001_Users.xml			
Import U	sers Imbedded	Interface Point				
-Sign on S	ettings					
End Users must Signon to this Framework			in both MS-Windows and Web Browser Applications 💟			
() Users	Sign on Locall	y to Use the Framework			Maximum Signon Attempts All	owed
User Can Change Own Password				←If Maximum Allowed Sign on Atte		Attempts Exceeded
	 O Users Sign on to a Remote Server to Use the Framework ✓ Users May Work Offline if the Remote Server Is Not Available 				Advise User with a Mess Framework Fatal Error	age

Optionally Set up Framework Users and Security

You can optionally use Framework users and security:

• Display Framework properties. In the User Administration Settings tab select the *Use Framework Users and Authority* option. Also select the option *Store Users in DBMS tables VFPPF06/07*. Save and restart the Framework.

Help About Web Details Developer Preferences	- iSeries Server Developer Preferences - Web Server 2 User Administration Settings
Authority Settings ✓ Use Framework Users and Authority ✓ Store Users in DBMS Tables VFPPF06/7	
Name of User Set to be Used	SYSTEM
Import Users Imbedded Interface Point	

- In the Administration menu select the Users option.
- Specify the user profile details and their authorities. For more information use the context-sensitive help by pressing F1.

Optionally Configure newlook User Profile and Password in the Framework

Most commonly the newlook user profile and password are the same as the the user profile and password used for the Framework superserver connection. In this case do not specify the newlook user id and password because RAMP will default to these values when starting a newlook session.

However, if you are using Framework Users and Security and if the newlook user profile and password are different from the Framework user profile and password, you can specify your newlook *User Profile* and newlook *Password* in the Framework. Alternatively you can specify these details when connecting to newlook.

If you want to specify these details in the Framework:

- Start the Framework.
- In the Administration menu select the Users option to display the User Details tab.
- Select your user profile.
- In the *newlook User* and *newlook Password* fields, define the user profile and password you use to connect from newlook to your System i server, or use the special value *PROMPT in both fields.
- Close the dialog and save the Framework.

Users 📃 Users		
G. User Pr Caption	Identification Icons User Details Custom Properties Authorities	di la constante
S DCXUSER UserProfile 1	User Profile DCXUSER New Password Caption UserProfile 1	
	Newlook User DCXUSER Newlook Password ********	
	Temporary Directory on PC: C:\DOCUME~1\eeva\LOCALS~1\Temp\ Email Address:	
	Export Users Import Users from XML	Close

Starting the Framework on the Web

There are RAMP-specific options when starting the Framework on the web. See this section in the Framework guide: Web Application Start Options.

When Many Developers Work on the Same Application

When modernizing large applications, it may be necessary that several developers share the work.

Handle Multiple Framework Versions

Multiple Developers Using newlook

Script Naming Convention

Handle Multiple Framework Versions

See Framework Versions.

Multiple Developers Using newlook

When multiple developers are identifying screens and fields in newlook, it is recommended that the developers work from local copies of the SID file with updates being merged into a central repository. The merge process should be performed by a single developer who is responsible for managing potential conflicts.

However, it is also possible to have multiple developers working on the same SID file via the network. You should note that working on a SID via a network is not as fast as working on a SID file locally and, as with all shared databases, there is always the risk of data corruption due to PC lockups or power outages. To work on a shared .sid file:

- Copy the newlook.sid (or whatever your shared dynamic repository is named) to a shared folder.
- In newlook choose the *Settings* option from the *Tools* menu, then click the *Settings* button in the *Dynamic Recognition Repository* section.
- In the *Shared* field specify the newlook.sid file on the shared folder.

Repeat the last two steps for all developers who want to work on the repository.

Many developers can simultaneously work on the same newlook.sid file. If developer A makes some customizations to a screen and saves the changes, developer B will see those customizations instantly. If two developers try to change the same object (i.e. screen, macro, etc) at the same time, newlook tells the second developer that the relevant object is currently locked by another developer.

Script Naming Convention

RAMP scripts are assigned names like INVOKE_SCRIPT_2, BUTTON_SCRIPT_7, etc.

The name reflects their purpose and the numerical suffix makes them unique within the current Framework, but they have no real programmatic purpose.

Where multiple developers are working on independent Frameworks with an intention to merge their work together at some later date, the possibility of duplicated script names exists. While this situation does not present a technical problem for RAMP, it can be confusing for developers trying to identify unique scripts.

Developers can change the names of the scripts in the Script Details area. The recommend way to do this is to append a short suffix to the generated script name, possibly relating to the 5250 screen or application that the script is associated with.

Also see xxxxxx is an orphan script and should be deleted.

Starting RAMP

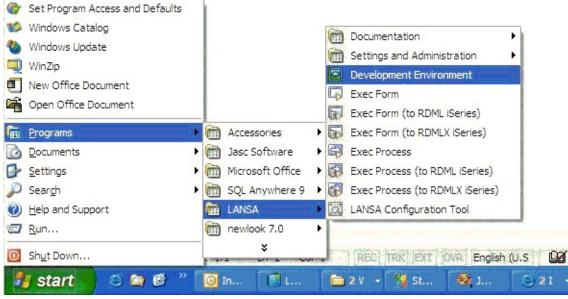
This section summarizes how you start LANSA and the features inside LANSA you will need when modernizing an application.

Start LANSA Start the Framework Start RAMP Start newlook Start the Instant Prototyping Assistant Start the Program Coding Assistant

Start LANSA

To start LANSA:

- Use the Start menu and display the Programs folder.
- Select LANSA.
- Select the Development Environment option

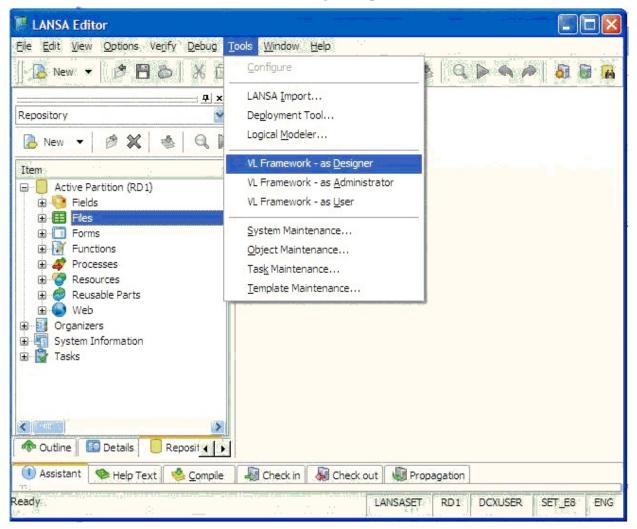


The LANSA development environment is displayed

Start the Framework

You start the Framework from the LANSA development environment:

- Display the Tools menu.
- Select the VL Framework as Designer option.



Start RAMP

You start RAMP from the Framework window:

- Display the Framework menu.
- Select the RAMP Tools... option.

Demo Application	
<u>File Edit View Actions Tools H</u> elp	(Framework) (Administration)
🎽 Sign Off 🕘 New 💌 🔗	(New)
Demo Application On Tool Bar Demo Application HR	(Properties) (Applications) (Commands) (Menus)
	(Design Code Tables) (Program Coding Assistant) (Instant Prototyping Assistant)
	(Instant Prototyping Assistant) (RAMP Tools) (Save) (Save As) (Save and Restart) (Save and Exit)
	(Execute as Web Application) (Web Consoles) (Assistance) (Tracing)
	Messages Local ENG DCXUSER 3/22/06 15:03

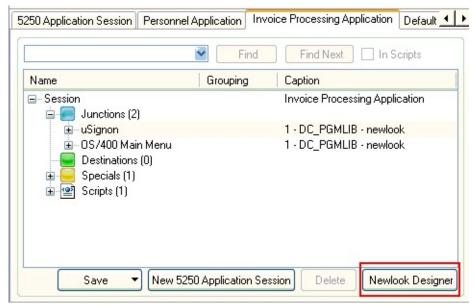
The RAMP and newlook Tools window is displayed.

Start newlook

• You start newlook in the RAMP Window There are two different ways you use newlook:

Identifying Screens

To identify 5250 screens, click on the newlook Designer button on the bottom of the RAMP and newlook tools window:



The newlook window is displayed:

	1 -	Off	fline	- ne	wlook	5									X
ii E	<u>F</u> ile	<u>S</u> es	sion	<u>E</u> dit	<u>V</u> iew	Macros	s <u>T</u> ools	Fund	tion	<u>K</u> eyb	oard			»	2
1	Í	S.	6	¥	te f	10) 🖭		Ŗ	Ø	2,2	٩	#	»
															_
	8	.	X							ENU	OVR	3	2 4	⊳	

Use the Session menu to establish a connection to the newlook server.

Defining Screens

After you have identified the screens using newlook Tools, you need to define the screens in the Framework and track the navigation between them.

To do this, start the newlook emulator session by clicking on the message newlook has not been started in the message area:

```
Click on any messages below to see available actions
```

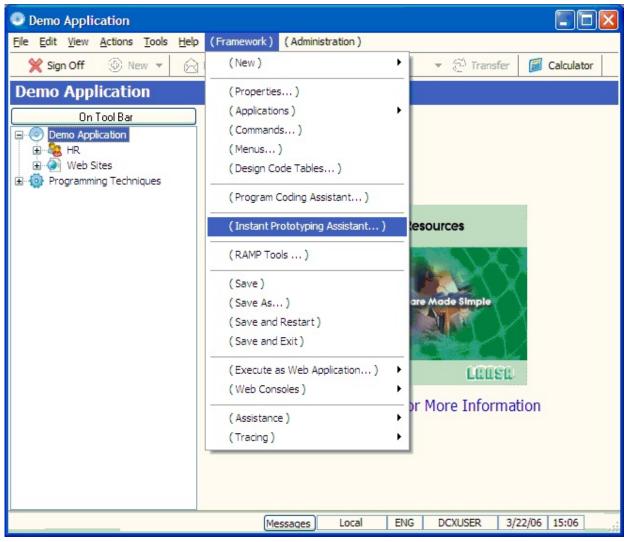
The newlook Emulator Session is started in the RAMP window. Use the Session menu to connect to the server.

Start the Instant Prototyping Assistant

The tutorial movie Create a prototype of your application - 3 minutes shows how to use the Instant Prototyping Assistant.

Use the Instant Prototyping Assistant to quickly prototype your application or to modify an existing prototype

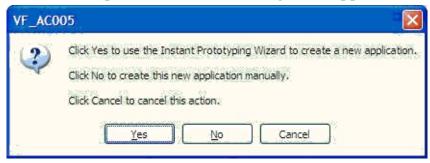
To start the Instant Prototyping Assistant, use the Instant Prototyping Assistant... option in the Framework menu.



Alternatively, select the New Application or New Business Object options from the popup menu in the navigation pane:

🥦 Human Relations
Eile Edit View Actions Tools Help (Framework
🐂 💥 Sign Off 👘 😰 New 🗶 🔤 Email 👘 🗁
Human Relations
On Tool Bar
Demo Application Programming Techniques (New Application)
(New Business Object)
(Properties)
(Delete)
Position 🕨

And then respond Yes to the message that appears:



The Instant Prototyping Assistant is displayed:

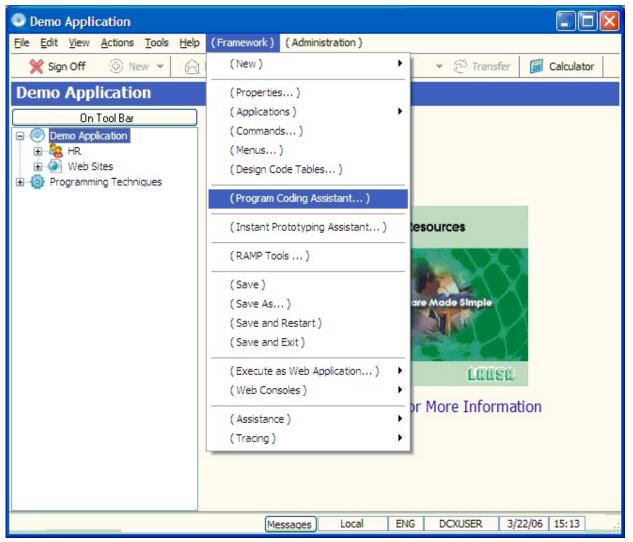
🔲 Instant Prototype Assistant 🛛 🔀					
define business objects attach actions					
What actions can users do with "business objects" ?					
Step 2. Enter the names of all the actions below: (separated by commas)					
Windows designs use the Object->Action approach. - (i.e. select the object you want to work with, and then choose what you want to do with it.) The actions should be described in end user terms, not in IT terms.					
 Very concise words are used to describe "actions", because the object being worked with is already knowr If you select an object in MS-Powerpoint and use the right mouse there is a concise menu option "Copy". It does not say "Copy this text box to the Clipboard". Short verbs tend to be used to describe actions. (e.g. Copy, New, Edit, Print, Approve, Transfer, Reply, Renew) Short nouns are also used to refer to things that directly relate to the business object. (e.g. Details, History, Charges, Claims, Attachments, Schedule, Contacts, Documents, Expenses) For example: If there is a "business object" called Customers, you could do these things with a Customer: 					
Actions: Details , New , Notes ,					
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.					
Image: Strate of the strate					
Image: Notes Image: Description of the second s					
<< Back Next >> Cancel					

Start the Program Coding Assistant

The tutorial movie Create a filter and snap it in - 4 minutes shows how to use the Program Coding Assistant.

Use the Program Coding Assistant to quickly create the code for Framework filters and RAMP screens.

To start it, use the Program Coding Assistant option in the Framework menu:



Concepts

Steps Involved in Using RAMP Framework Window RAMP Window Types of Screens OBJECT-ACTION User Interfaces

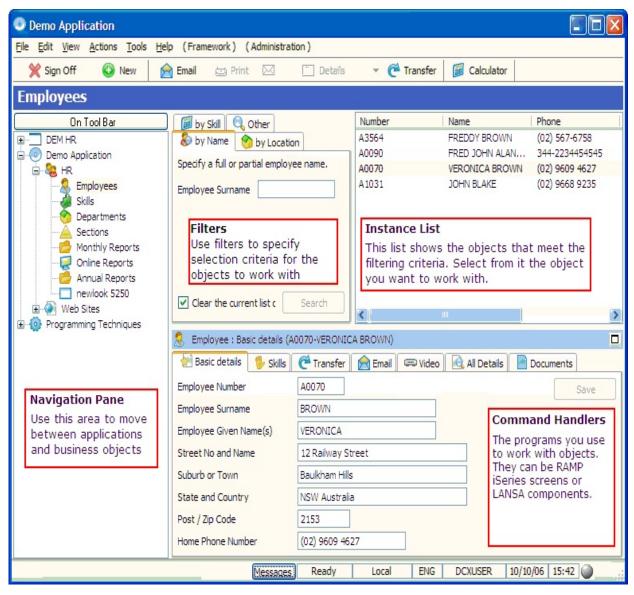
Steps Involved in Using RAMP

You need to complete these steps:

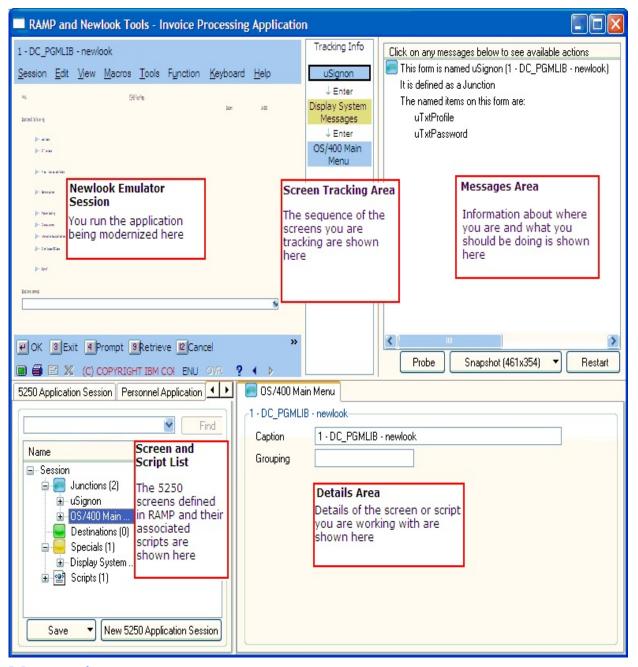
Step	Comments	Navigation Modernization	Application Modernization
1. Create a prototype of your application.	The prototype will evolve into the final application. You need access to the subject matter expert at least during this stage.		
2. Identify the 5250 screens in the existing application.	You use the newlook Designer for this.	Identify only entry point 5250 screens	Identify all 5250 screens that need to be modernized as well as significant fields
3. Record the 5250 entry point screens and snap them into the Framework.	The Framework needs to know how to access and display the screens.		
4. Create the	Using	Not necessary.	

required Framework filters	powerful filters is the basis of reusing the 5250 screens in new modernized ways.	
5. Optionally add new features making use of Windows functionality	For example you may want to add advanced screens for for instance email, video, graphing.	
6. Deploy	Deploy your application	

Framework Window



RAMP Window



Message Area Screen Tracking Area newlook Emulator Session Screen and Script List Details Area

Message Area

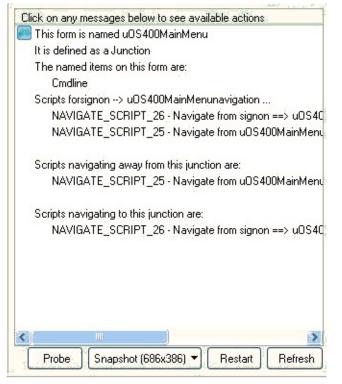
RAMP Window	Screen Tracking Area	newlook Emulator Session	Screen and Script List	Details Area
----------------	----------------------------	--------------------------------	---------------------------	-----------------

The RAMP message area shows messages about where you are and what you should be doing.

When newlook is running, messages are shown for the screen selected in the Screen Tracking Area.

If messages have a message icon, click on it to see what actions are available.

The message area has buttons you can use when tracking screens:



Probe	Use the Probe button to examine the layout of the current 5250 screen and produce an online report. It is used for problem analysis and to determine the rows and columns used in a subfile.
Snapshot	Use the SnapShot button to take a snapshot of the current 5250 form in GUI or 5250 mode and save it as bitmap. These images:

	• Can be dragged and dropped onto RAD-PAD prototype command tabs to enhance communications during design sessions with other developers or end-users.
	• Are useful for producing system documentation
	• Are an aid to remembering exactly what 5250 screen is associated with a junction, destination or special screen.
	The (nnn x nnn) numbers on the button indicate the pixel size of the snapshot that will be saved. These numbers will change as you change the layout of the RAMP Tools window.
Restart	Use the Restart button to erase the tracking information and restart tracking.
Refresh	Use the Refresh button to force the RAMP choreographer to re-examine the current 5250 screen. This is useful in situations involving 5250 flash screens.

Screen Tracking Area

RAMP	Message	Emulator	Screen and Script	Details
Window	Area		List	Area
		Session		

The Tracking area displays the screens you have displayed in the current newlook 5250 emulator session. When you end the newlook session, the tracking information is cleared.

Colors used in the Tracking area for screens indicate their status and type you have assigned to the screen:

Unknown Form	The screen has not been identified in newlook. Before using RAMP tools, you must identify the screens of your application using newlook.
wrkoutg	The screen has been identified in newlook, but it has not been defined in RAMP.
	You need to define the screens in your application according to their purpose:
	• Destination screens are screens where the end-user works
	• Junction screens are used for navigation only
	• Special screens are used for messages etc.
	To define a screen, click on the message saying that the form has not been defined.
uSignon	The screen is a junction screen.
Display System Messages	The screen is a special screen.
wrkjobg	The screen is a destination screen.

newlook Emulator Session

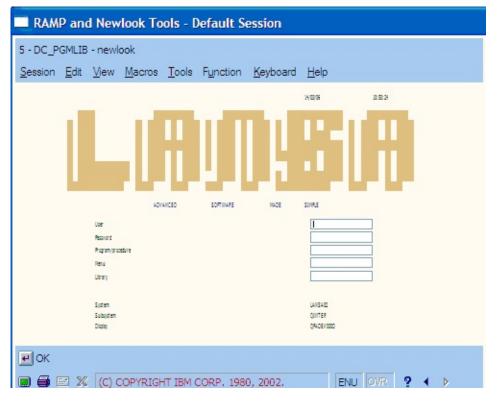
Alea Alea	RAMP	Message	Screen Tracking	Screen and	Details
	Window	Area	Area	Script List	Area

You use the newlook emulator session to run the application you are modernizing.

When newlook has not been started, the message newlook has not been started is shown in the message area:

Click on any messages below to see available actions	1
A Newlook is not started. Click on this line to start Newlook.	

Click on the message to start newlook. Then use the newlook Session menu to connect to the server:



Screen and Script List

RAMP Window	Message Area		newlook Emulator Session	Details Area
----------------	-----------------	--	--------------------------------	-----------------

The screen and script list shows all the 5250 screens defined in the Framework and the associated scripts:

250 Applica	tion Session Personnel Applica	ition Invoice Processing	Application Default Session
	× (Find Find Nex	kt 📃 In Scripts
Name	Grou	Last Changed	Caption
⊆Session ⊕ -	n Junctions (3) Destinations (1)		Default Session
	EnrolEmployee	20061025-145914-D	1 - DC_PGMLIB - newlook
	RETURN	20061025-150042-D	EnrolEmployee - Return to near
÷	BUTTON_S INVOKE_S Specials (1) Scripts (8)	20061025-150046-D 20061025-150048-D	EnrolEmployee - Handle functio EnrolEmployee - Invoke this for
	Save 🔻 New !	5250 Application Session	Delete Newlook Designe

Select the screens and scripts you want to work with.

You can use the Find field on the top of the list to locate screens and scripts. If you want to search the contents of scripts, tick the In Scripts check box.

There are two ways you can save your changes to the Framework in the RAMP window:

Full Save	Performs a full Framework save including the generation of all scripts for execution in end-user mode and the uploading of web server details.
Partial Save	Performs a partial Framework save so that your work is fully recoverable, but does not generate run-time scripts or upload server details. You will need to do a full Framework save to deploy your application or execute it in end-user mode.

You can use the New 5250 Application Session button to organize screens and scripts into distinct 5250 Application Sessions (see Organizing Screens and Scripts).

The newlook Designer button starts a newlook client session you need to use when identifying screens.

Organizing Screens and Scripts

If your applications are large and complex, you may want to divide the screens and the associated scripts into separate groups along application lines. You can do this by creating separate *5250 application sessions* for them in the RAMP window.

Developing applications with hundreds of screens becomes increasingly complex to manage because of the number of objects they contain. Also, the initial start up time of an application increases in a linear manner according to the number of objects it contains.

In this example three application sessions have been created in addition to the Default Session (5250 Application Session, Personnel Application and Invoice Processing Application):

5	250 Application Session	Personnel Application Inv	oice Processing Application	Default Session
[Find	Find Next 🔲 In So	cripts
	Name		Caption	
	<mark>⊟Sessio</mark> n		Invoice Processing	Application
	😑 📻 Junctions (2)			
	🖃 uSignon		1 - DC_PGMLIB - n	newlook 🔤
	😟 🔮 Scrip	ots		
	😑 - OS/400 Maii	n Menu	1 - DC_PGMLIB - n	iewlook
	😑 🔮 Scrip	ots		
	NAV	IGATE_SCRIPT_6	Navigate from uSig	non ==> 0S/400 Main M
	NAV	IGATE_SCRIPT_7	Navigate from OS/	400 Main Menu ==> uSig
	🛛 🥃 Destinations	(0)		
	😑 🥃 Specials (1)			×
		Save 🔻 New !	5250 Application Session	Delete Newlook Designer

Note

that the 5250 application sessions are completely independent of each other and have no knowledge of each other's existence. This means that a script in one session cannot navigate to an object in another application session and that you will most likely have to duplicate some common scripts such as logon and logoff and messages.

A separate newlook session will be started for each 5250 Application Session.

To create a new grouping, click on the New 5250 Application Session button in the RAMP window. You can edit its caption in the Session Details area.

Only one 5250 Application Session can be active at any time. To change the application session, simply display the tab for that session. All screens that you define and scripts you create are stored in the current 5250 Application Session.

Details Area

Session Details Destination Screen Details Script Details

Session Details

RAMP Window	Message Area			Screen and Script List
		Area	Session	

Use the Session Details to specify various settings for your 5250 Application Session:

-Session - Default Session								
Capti	Caption Default Session							
User	User Object Name / Type 5A5EC45BB24F41CDB98DB787065804A Verify Name							
Def	-Default RAMP Layout Dimensions-							
Hei	ght 380	Width 700	Тор		Left			
Тор	o Mask Height	Bo	ottom Mask Hei	ght	27			
	MP Screen Layou		C) Flow La	yout			
Scr	oll Bars Display H	orizontal Scroll Bars	; 🖪	Display	Vertical Scrol	l Bars		
V L	ock framework w	hen unknown 5250	form is display	ed				
🗖 R	euse existing cor	nnection's user prof	ile and/or pass	word				
Alwa	ys link this sessio	n to a server with L	Jser Object Nar	me / Type				
Spe	cial Field Handing)						
	5250 Field Nam	e Function Key	VL Handle	r (class VF	_AC017 obje	ect)		
1		F4						
2	2 F4							
3	3 F4							
4		F4						
5								

Caption	The caption of the RAMP 5250 Application Session.
Height	The default height of 5250 screens when displayed in the Framework.
Width	The default width of 5250 screens when displayed in the Framework.
Тор	The default distance between the top of the RAMP screen tab and the 5250 screen.

- 0	You can use this option to Hide screen titles in RAMP Screens
Left	The default left indentation of the 5250 screen when displayed in the Framework.
Top Mask Height	The default height of a mask you can use to hide the top of the 5250 screen.
	You can use this option to Hide screen titles in RAMP Screens
	This option is not applicable to RAMP Web.
Bottom Mask Height	The default height of a mask you can use to hide the bottom of the 5250 screen.
RAMP Screen Layout	If RAMP Screen Layout Style is set to <i>Flow</i> , RAMP screens will be automatically resized to fit into the space available to display them.
Style	If <i>Flow</i> is used:
	• Specific positioning and sizing of screens is not supported,
	• Top and bottom masking of screen areas cannot be used to hide screen content.
	• You cannot use or show the function key blue bar.
	• Display Horizontal Scroll Bars and Display Vertical Scroll Bars options cannot be used for the obvious reasons.
	<i>Fixed</i> means the RAMP screens are not resized to fit into the space available to display them.
	Note that you can override this setting for individual destination screens.
Scroll Bars	If the <i>Display Horizontal Scroll Bars</i> option is checked, VLF.WIN applications will display horizontal scroll bars when a Fixed size 5250 screen will not fit in the display area. VLF- WEB/NET applications always act as if the <i>Display Horizontal</i> <i>Scroll Bars</i> option is checked.
	If the <i>Display Vertical Scroll Bars</i> option is checked, VLF.WIN applications will display vertical scroll bars when a Fixed sized 5250 screen will not fit in the display area. VLF-WEB/NET

	applications always act as if the <i>Display Vertical Scroll Bars</i> option is checked.
Lock Framework when unknown 5250 form is displayed	 This option applies a lock to the Framework when an unknown 5250 screen is encountered. When a lock is applied, the user cannot move around within the Framework until they navigate to a defined 5250 screen. They can exit from (for example, shut down) the Framework when such a lock has been applied. Typically this option is used to trap unknown and/or unexpected 5250 screens. In highly defined and managed sessions, where every 5250 screen should have been defined to RAMP, set this option on. In unmanaged sessions always set this option off.
Reuse existing connections user profile and/or password	Use this option to indicate that when this 5250 application session needs to connect to a server it should reuse the same user profile and/or password details as were used to establish the last successful server connection. This option may be used to prevent the user from being prompted to input their user profile and/or password repeatedly for each new 5250 application session that needs to be started. Typically they are only prompted for the first application session they establish. This option may be automatically overridden by individual user profile options or by super-server connection values. The Framework remembers the last user profile and/or password used to establish a server connection only until the user exits from the Framework, at which point the details are lost.
Always link this session to a server with User Object Name / Type	Normally when a user needs to connect a 5250 application session they will be asked to choose which server they want to connect to. Use this option to prevent the user from having to, or being allowed to, making this server connection choice. Using it unconditionally links a 5250 application session with a server.

To use this option first assign an unique *User Object Name / Type* to the server.

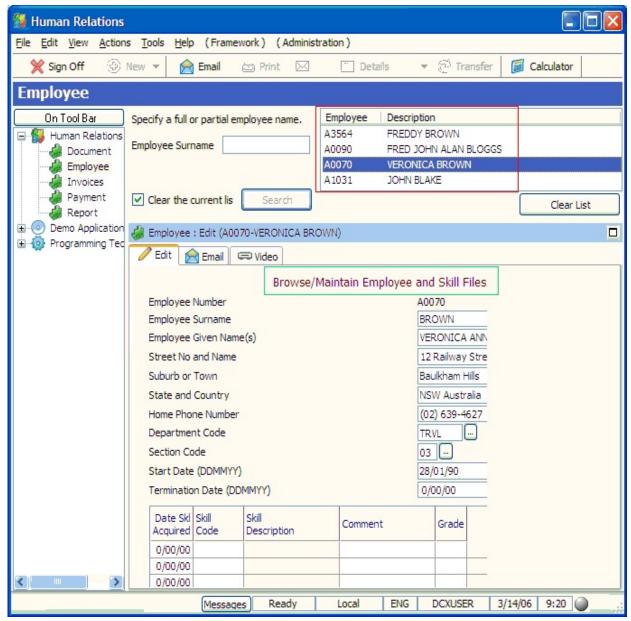
Use the Framework Administration menu Servers option to do this.

For example, this server has been assigned the User Object Name / Type SERVER_2.

			1	100 million (100 m		
	Servers					
	Sequence 1. E Server	Server Details Identification	Icons	Contraction of the local division of the loc		
	ad 1 LANSA02	Caption	LANSA02	(ENG)		
		Hint:		(BNG)		
		Sequence:	1			
		Internal Identifiers	SERVER_2			
		Unique Identifier:	400			
		User Object Name / Type	SERVER 2	Verify Name		
		Restricted Access				
	New Delete	Last Changed				
	L MEN		LANSA	Comms. Admin Close		
	Next, set the 51	50 application	n session to use the	e same name (eg:		
	SERVER_2).					
		ne 5250 application session and the server with user objec				
	name/type SERVER_2 are unconditionally linked.					
	The user can no	ser can no longer choose which server to associate the				
	5250 application session with.					
	To remove this	option from a	session set it back	to the default		
	value of blank.					
1						
Special	Advanced pron	npting facility	for fields.			
Field						
Handling	You specify the name of the field to be prompted, the function					
Tanuning	key to be used and the Visual LANSA form that is used as the					
	prompter.					
	For more infor	mation refer to	Advanced Promp	nting		
				5		

Hide screen titles in RAMP Screens

In most cases 5250 screen titles are redundant in RAMP screens because the navigation elements in the Framework Window clearly indicate the object being worked with and the command being executed:



Therefore RAMP screens look more natural without titles in the Framework:

Human Relations			
File Edit View Actions			
		1	Calculator
	New 💌 🔂 Email 🖾 Print 🖂	Details + (
Employee			
On Tool Bar	Specify a full or partial employee name.	Employee Description	
🖃 🎒 Human Relations	Employee Surname	A3564 FREDDY BROWN	
Document	Linployee Sumane	A0090 FRED JOHN ALAN BLOGGS	
Employee		A0070 VERONICA BROWN A1031 JOHN BLAKE	
Payment	Clear the current lis Search	A1051 SOUNDERRE	
Report	Clear the current lis Search		Clear List
🗉 💿 Demo Application	THE LIDUVEE , LUIL (AUD/D-VERONICA DR	OWN)	
🖻 🍥 Programming Tec	Cedit Remail Con Video		
	Employee Number	A0070	
	Employee Surname	BROWN	
	Employee Given Name(s)	VERONICA ANN	
	Street No and Name	12 Railway Stre	
	Suburb or Town	Baulkham Hills	
	State and Country	NSW Australia	
	Home Phone Number	(02) 639-4627	
	Department Code	TRVL	
	Section Code	03 📖	
	Start Date (DDMMYY)	28/01/90	
	Termination Date (DDMMYY)	0/00/00	
	Date Skl Skill Skill Acquired Code Description	Comment Grade	
	0/00/00		
	0/00/00		
	0/00/00		
< N	0/00/00		
	Messages Ready	Local ENG DCXUSER 3/1	4/06 9:23
	C. CHARLEN CO. C.		

There are Two Ways to Hide the Title.

Two Ways to Hide the Title

You can hide the screen title either by moving the RAMP screen up so that the title is hidden or by applying a mask on the title to hide it.

You set the RAMP screen position and mask in the Session details of the Default Session in the RAMP Tools window:

RAMP and Newlook Tools - Default Session	
	Click on any messages below to see available actions Newlook is not started. Click on this line to start Newlook
Session - Default Session Name Session Junctions (6) Destinations (6) Specials (1) Scripts (27)	Left
	ller (class VF_AC017 object)

You can override these settings for individual destination screens by changing the Layout Dimensions in the Destination Screen Details.

Moving the Screen

To move the screen up so that the title is hidden, set the Top property to a negative value:

Caption Defai Heig	Default Session Default Session Default Session Ut NewLook Layout I Jht 380 W Mask Height	/idth 700	Top 25- Left	
2	ock framework when i sial Field Handing 5250 Field Name	1 122 02 0 0 0 0	is displayed VL Handler (class VF_AC017 object)	
1	5250 Field Name	Function Key	VL Handler (class VF_ACUT7 object)	
2		F4		
3		F4		
4		F4		
3		··		

Masking the Title

To mask the title, set the Top Mask Height property to a height that covers the title:

tion Default Session		
efault NewLook Layo Ieight 380 Fop Mask Height	Width 700	Top Left
Lock framework whe	n unknown 5250 forn	n is displayed
pecial Field Handing		
pecial Field Handing 5250 Field Nam	e Function Key	VL Handler (class VF_AC017 object)
	e Function Key	VL Handler (class VF_AC017 object)
5250 Field Nam		VL Handler (class VF_AC017 object)
5250 Field Nam	F4	VL Handler (class VF_AC017 object)
5250 Field Nam	F4 F4	VL Handler (class VF_AC017 object)

Destination Screen Details

RAMP	Message	Screen	newlook	Screen and
Window	Area	Tracking	Emulator	Script List
		Area	Session	

When a Destination Screen is selected in the Screen and Script List, the details of the destination screen are shown:

	Employee ISA02 - newl	ook					
aption	aption 1 - TestLANSA02 - newlook						
ouping					_		
	AMP Layout I	Dimensior	ns				
	380	Width	700	Тор		Left	
Top Mask	Height		Bot	tom Mask He	igh	t 27	
RAMP Scr	een Layout S	5tyle					
	Session			🔘 Fixe	ed L	ayout 📀 Flow Layout	
Function I	Key Enablem	ent			-1	-Associated Command Handler(s)	
	Caption		Enable	Seg 🔊		Link to Command (Tab)	in O 🔨
Key Clear	Clear			Seq ^		About Framework	New
Enter	DAccor			2			New
	Page Up			3		Assistant Example 1	New
	Page Down			4		Assistant Example 2	New
Print	Print			5		Assistant Example 3	New
Help	Help			6		Spooled Files	New
F1	Help		~	7		About	Proc
F2	F2			8			The
F3	Exit			9			The 🗸
F4	Prompt			10			
F5	F5			11			
	F6 F7			12		Refresh	
F6				13			
го F7 F8	F8			14		Session Id *AUTO	-

You can specify these details for the destination screen:

Grouping	Optionally type a grouping name for this screen.
	You can use this option to enter the same grouping name to related screens so that they can be sorted together in the Screen and Script List.

	For more fundamental organization of screens and scripts, see Organizing Screens and Scripts.
Default RAMP Layout Dimensions	Use these properties if you want to permanently override the default layout dimensions set in Session Details for this screen.
RAMP Screen Layout Style	If RAMP Screen Layout Style is set to <i>Flow</i> , this screen will be automatically resized to fit into the space available to display it. If <i>Flow</i> is used:
	• Specific positioning and sizing of the screen is not supported,
	• Top and bottom masking of the screen area cannot be used to hide screen content.
	• You cannot use or show the function key blue bar.
	• Display Horizontal Scroll Bars and Display Vertical Scroll Bars options cannot be used for the obvious reasons.
	<i>Fixed</i> means the RAMP screen is not resized to fit into the space available to display it. <i>Session</i> means the value is inherited from the Session's properties.
Function Key Enablement	This is a list of all the available function keys in 5250 screens.
	You can use the list to enable or disable function keys in the 5250 screen and also to enable or disable the runtime appearance of push buttons in the RAMP screen that have the same functionality as the corresponding function key. By default, when a screen is defined as a
	destination, all function keys are disabled and

	the corresponding buttons are enabled. This means that when the screen appears, pressing the function key will have no effect, but a corresponding button will appear on the RAMP screen which is functionally equivalent the function key in the original 5250 screen. Note that function key enabling is only valid for those function keys already present in the 5250 screen.
	For example, if a 5250 screen is designed to have function keys F1, F3, F6 and F12, enabling the F10 key will have no effect in the application since that key has no functionality in the original screen. However, you can still enable the F10 in the RAMP screen if you add your own script for it in the button script of the destination screen.
	 To enable a function key, tick the check box in the Enable 5250 column. To display the function key as a button, tick the check box in the Enable VLF
	 column. The captions of the buttons can be changed in the Caption column.
Associated Command Handlers	The command handler tab where the RAMP screen will be attached. The command handler tabs are created when you prototype your application.
Session	Specifies what System i 5250 session (ie: job) should be started for the screen. <u>*AUTO</u> : is the default value and indicates that the Framework should manage the required 5250 session(s) automatically. This type of session is a <i>managed</i> session. It is

fully integrated with the Framework, applications, business objects and instance lists and all scripting facilities are available. <u>SESSION_A -> SESSION_Z</u>: allow you to specify that an *unmanaged* session is to be started for the command handler or tab. Unmanaged sessions are primarily used to log the user on and then drive them to a specific starting point. From that point forward the user can move around inside the 5250 application in an unmanaged way. Since the session is unmanaged only very limited scripting capabilities exist. For example, a script in an unmanaged session can not access the business object instance list. Equally, when a user returns to an active command handler / tab that uses an unmanaged session it is simply redisplayed as it was when they last left it. No attempt to navigate them or execute any scripts is attempted (because it is unmanaged).

Unmanaged sessions are useful because they allow large pieces of an existing application to be reused in the Framework very rapidly.

For example, an unmanaged session might be used as the only command associated with a business object named "System Tables". When the user clicks on "System Tables" in the Framework menu, a full screen 5250 session appears that logs the user on and then drives them to the 5250 menu that manages the maintenance of 50 (say) system tables. The entire "System Tables" facility composed of hundreds of 5250 screens (say) are now accessible in an unmanaged fashion, without the need to identify and enroll them in the Framework. If the users goes away from the "System Tables" tab and then come back again later the current 5250 session screen, whatever it is, is just redisplayed. No attempt is made to navigate the screen (ie: manage it) because in all likelihood they will have left it on an undefined or unknown 5250 screen.

In short, you should always use *AUTO unless you have a specific need to log a user on, drive them a defined starting point in the application, and then allow them to move around wherever they like within the 5250 application area.

NOTE: When changing the session option ensure that you *select* associated command handler by clicking on it.

This command handler is not correctly selected:

Email Employees

and changes to the session will be ignored. This command handler is correctly selected:

🗹 Email 🛛 Employees

and changes to the session will be recorded.

You need to do this because sometimes a single destination screen is associated with multiple command handlers which can have different sessions, so you need to positively indicate the one you wish to work with.

Script Details

RAMP	Message	Screen	Emulator	Screen and
Window	Area	Tracking		Script List
		Area	Session	

The Details area shows the details of the script selected in the Screen and Script List.

The scripts are most often generated automatically as you trace your application. Sometimes it is necessary to edit the scripts.

You can use the Using the Scripting Pop-up Menu to help you to format and edit your scripts.

RETURN_SCRIPT_4	
Role : Return from 1 - DC_PGMLIB - newlook	k to nearest junction
Script RETURN_SCRIPT_4	
Caption EnrolEmployee - Return to r	nearest junction form
Grouping	
/* Navigate back to the ne	earest junction (PersonnelSystemMainMenu) */
/* Send the key required	Cu <u>t</u> <u>C</u> opy stemMainMenu */
SENDKEY(KeyF12);	Paste Undo
	Redo
	Upper Case
	Lower Case
	Larger font
	Smaller font
	Show Line Numbers
	Current Framework
	Current Application
	Current Business Object
	Current Command
	Current Instance List Entry
	5250 Subfile Handling
	Session Control
	Function Key overrides
	SENDKEY - Common Function Keys 🕨
	Common RAMP Script Functions
< [
Commit Changes Check Script	Use Tracking Info to prototype a script to Return from 1 - DC_PG

Types of Screens

Classifying the screens in your 5250 application is the starting point in modernizing your application:

- A Destination Screen is the 5250 screen where the end-user performs actual work. These screens are snapped into the Visual LANSA Framework without any modification.
- A Junction Screen is used for navigation only. They are hidden in your modernized application.
- A Special Screen is a messages or other similar screen that does not fit the above two categories.

Destination Screen

A destination screen is a screen in which the end-user works with an object.

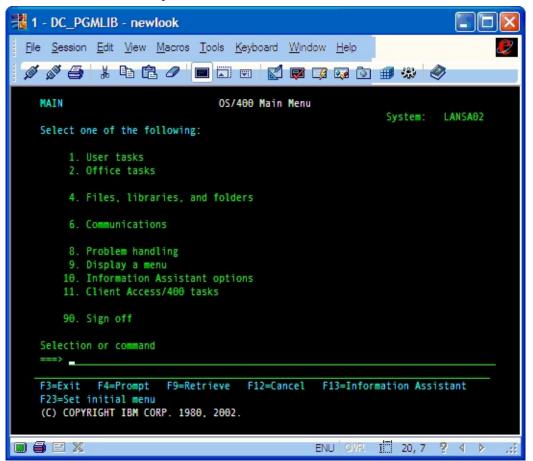
📲 1 - DC_PGMLIB - newloo	bk		
<u>File S</u> ession <u>E</u> dit <u>V</u> iew <u>M</u> a	icros <u>T</u> ools <u>K</u> eyboard	<u>W</u> indow <u>H</u> elp	2
Ø Ø Ø & B B &	/ 🔳 🖾 💌 🛃	👿 🗔 🗔 🗄	• * Ø
Brow	se/Maintain Employee	and Skill Files	
Employee Number Employee Surname Employee Given Name(s) Street No and Name Suburb or Town State and Country Home Phone Number Department Code Section Code Start Date (DDMMYY) . Termination Date (DDMMY	11/1 11/1	ACKSON ICHAEL PETER Melissa Place est Pennant Hills W Australia 71-7773 M + 1 + 1 + 1 /01/51 +	
Date Skl Skill Acquired Code 0/00/00 0/00/00 0/00/00 0/00/00		Comment	Grade
F1=Help F3=Exit F4=P	rompt F12=Cancel F1	14=Msgs	
• = • ×		ENU OVR I	■ 4,43 ? < ▷ .;;

Destination screens can be reused without any modification in RAMP applications:

👹 Staff Management				[
Eile Edit View Actions Tools Help (Frame	ework) (Administr	ation)			
🎇 Sign Off 🛛 😥 New 👻 🚱 Email		Details	* 澄市	ansfer	
Employee					
Personnel	Employee	Description			· 🔺 📉
Employee Number	A 1020	DOUGLAS ADAM	1 PETER		_
	A1234	JACKSON STEPH	HEN		
	A1001	JONES BEN			~
Search	4	Clea	ar List]
di Employee : Edit (A1234-JACKSON STEPHEN)					
All a second sec			ć		
Browse/Main	ntain Employee	and Skill Fil	es	OK	
Employee Number		A1234		Help	
Employee Surname		JACKSON			
Employee Given Name(s)		MICHAEL		Exit	
Street No and Name		6 Melissa Pl	ace	Prompt	
Suburb or Town		West Pennant	Hills	Cancel	
State and Country		NSW Australi	a	Cancel	
Home Phone Number		871-7773		Msgs	
Department Code		ADM		× 1	
Section Code		04			
Start Date (DDMMYY)		1/01/51	11		
Termination Date (DDMMYY)		0/00/00 -			
renamación bace (bbinnin)		0,00,00	- Contraction of the second se		
Date Skl Skill Skill Acquired Code Description	Com	ent	Gra		
0/00/00					
0/00/00					
Messages Re	ady Local	ENG DCXL	JSER 1/	31/06 15:04	Ð. "

Junction Screen

The end-user uses a junction screen to move to destination screens.



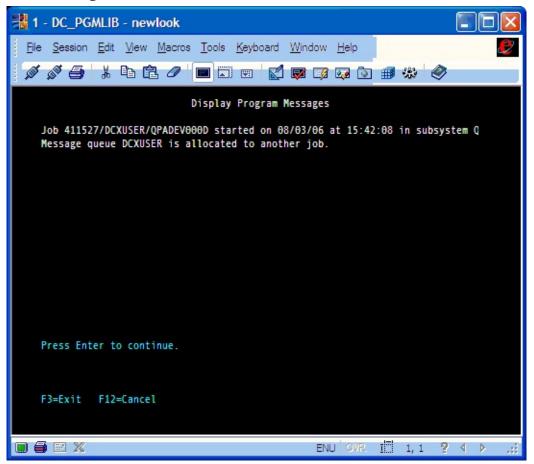
These navigation-only 5250 screens a become invisible to the end-user in a RAMP application.

Typical junctions are:

- 5250 menus
- 5250 "work with" style screens, which are really just data driven menus.
- Most 5250 screens where keys such as order numbers, customer numbers, product numbers etc. are entered to display or action detailed information.

Special Screen

Special screens are message and other screens that do not fit in the category of either navigation or destination screens.



These screens may appear unexpectedly at anytime in a 5250 screen flow. For example:

- The 5250 display message screen that appears at sign-on time
- The 5250 break message screen that may appear at any time
- Fatal error message screen(s) in your own applications.
- The 5250 resume interactive session screen.

Special screens usually have a script associated with them. The script is called an *elimination script* because this type of script usually sends a key or performs an action so as to eliminate the screen from the 5250 screen flow.

See Types of Scripts in RAMP for more information about elimination scripts.

OBJECT-ACTION User Interfaces

System i and Windows applications, including the Framework, share the same basic design for user interaction: Object-Action interfaces.

In these interfaces the user first selects and object and then the action to be performed on the object, as opposed to Action-Object interfaces (such as command line applications) where the command is specified first and its target object second.

Because of this fundamental similarity, System i applications fit naturally in the Framework model:

The navigation screens of a System i application are replaced by graphical elements in the Framework, such filters and instance lists, which the user can use to quickly locate the object they want to work with.

The options and associated screens in a typical Work with screen become a set of command tabs.

In the Framework the Object-Action model is expressed as a powerful graphical user interface (GUI).

System i and Framework Applications Share the Basic Model

The basic Framework concepts of business objects, filters and command handlers (screens) can be visualized in a System i application like this:

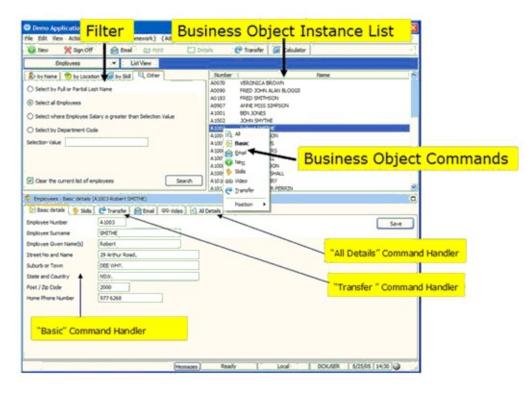
(*)Emmed (19+25) (*)EX (*)Seconds also days (*) (*)E(*)A(*)B(*)A(*)A(*)A(*)A(*)A(*) Berk with Opject Linns (Type choices, press Enter.		8		
Object	TNV. +NIME. +NASE +PRV. +USER. +ALL	ilter		
Business Object Insta Porters Particles Prestory:	ance List 055 7 fanter. farRemove 5=Disptay directory	ect Links	CODject Com	
Goug	DIR DIR DIR DIR DIR DIR DIR	2.0 2.0		C.P.B. C.L.C.M.
Parameters or cos stab P3#Exit F4=Prov F2=Display entir HPM a pP Not-feetersday.ity-itered	pen	fen nam	Bigard run Figure Same Anter engine in an anti- distance engine in an anti- distance engine in an anti- distance in anti	"Display Attributes"
e speciel Fistorie Fistorie	"Rename" Comman	for the second se	n Prass debr is vertime Fisher filstampt filst	Command Handler

Here you have a:

Filter	Where the Work with command provides you with options to filter the list of objects that are displayed. (Many "Work with xxxx" interfaces allow you to filter inside the main display as well).
Business Object Instance List	The list of links that match your filter's search criteria. These links are your business objects.
Business Object Commands	The Options such as 2=Edit, 7=Rename, 8=Display that you can execute against an individual business object.
Command Handlers	The programs that execute when you execute a command (7=Rename or 8=Display attributes examples are shown).

In the Framework, the same concepts are visualized as a graphical user interface

(GUI) like this:



Modernization Issues

The most important and complex 5250 program in an application can become a modernization trap

How long will it take to RAMP my application?

The most important and complex 5250 program in an application can become a modernization trap

The biggest and meanest modernization trap involves the most important and usually most complex 5250 program in an application. In an ERP application this program handles Order Entry, in an Insurance application it is the Policy Master Update.

Every 5250 application has at least one of these big and mean 5250 programs.

It is attractive and logical to involve this type 5250 program in any modernization proof-of-concept exercise on the simple basis that "if RAMP can handle this program then it can handle anything".

As a result a lot of time may be spent understanding the peculiarities of this program and scripting for them. This okay ... *unless* handling it consumes excessive amounts of time and diverts all attentions away from the hundreds (or thousands) of other important 5250 programs that also need to be modernized. In this case it can become a trap.

- *An ISV site should consider*: Which program would be the very first one you would change to a new Visual LANSA component so as to best show off your modernized product to potential customers?
- An in-house development site should consider: Which program would the end-users gain the highest productivity and usability improvements from if it was changed to a new Visual LANSA component? What program, if it was replaced by something better, would garner the most management and end-user support for the modernization project?

The answer in both cases is quite probably *the biggest and meanest 5250 program*.

Why not consider replacing it with something better?

If this is true, then the next question should be: "Why are we spending all this time and effort trying to reuse it, instead of just starting to replace it with something better?"

The reason is obviously to avoid the time and cost involved in replacing it.

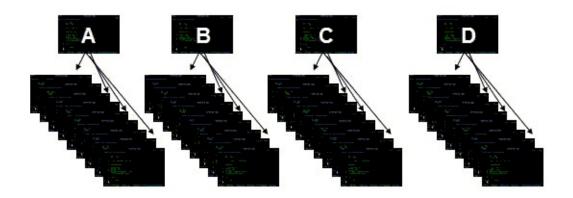
However, if the commercial reality is that for various marketing, business and political reasons it will need to be replaced sooner rather than later, you should seriously consider doing it now, instead of spending an unreasonable amount of time trying to reuse it and allowing it to become the complete center of attention to the detriment of all the other 5250 programs that also need to be modernized.

How long will it take to RAMP my application?

Important Note: This answer refers to RAMP stage 2 only - reusing your existing 5250 screens. It has *nothing to do with RAMP stage 3* - replacing your 5250 screens with Visual LANSA components.

It depends on the approach you use.

Imagine a simple 5250 application made up of four menus (or some other common access points) and 36 other screens like this:



We recommend you use this approach:

1. Initially Perform a Rapid Navigation Modernization

In this example you would identify and define the four menus (or access points) A, B, C and D only, and snap them into RAMP as full screen destinations.

The *entire* 5250 *application*, with its modernized navigation, *could now be deployed to your end users*.

Normally you would also fully modernize at least some part of the application itself, to add more value to it.

At this stage answering the question "*How long will it take to RAMP my application*?" is easy: Allow 15 minutes per menu (or common access point). So for this example, allow $4 \ge 15$ minutes = **1 hour**.

2. Now Perform Selective and Incremental Application Modernization

Now assess application areas A, B, C and D:

- How frequently are they used?
- Will full modernization increase end user productivity? How? What needs to be done?
- Will full modernization improve the end user experience? How? What needs to be done?
- Will full modernization aid the demonstration and marketing of your product? How? What needs to be done?

Based on these assessments you might decide to:

- Modernize application area A and deliver it to your users as an initial release.
- Later modernize 60% of application area C and deliver it to your users as a new version.
- Not fully modernize application area D at all, because it does not add business value.
- Finally, modernize 25% of application area B and deliver a final version to your users.

So answering the question "*How long will it take to RAMP my application?*" depends upon how you approach this step.

The question cannot be answered until you decide what parts need to be fully modernized, how much work needs to be done, and in what order.

Key Points

- Navigation modernization is very rapid.
- Application modernization takes longer, but adds significantly more value.
- You can deliver a modernized 5250 application incrementally. You don't have to do it all in one go.
- You are not forced to fully modernize all of a 5250 application just to use it in RAMP.
- Some parts of an application may never be fully modernized before they are replaced with new Visual LANSA components instead.

Tutorials

There are two very different ways of modernizing an application with RAMP:

- The steps described in the tutorial Modernizing a Complete Application is the most appropriate way of modernizing most applications.
- The alternative way is Modernizing Application Navigation. This approach is fast, but the resulting application does not make use of all the powerful features provided by the Framework user interface.

Scripting Tutorials show you how to manage your 5250 screens in the modernized application.

Modernizing a Complete Application

This tutorial introduces the key concepts required to *modernize a complete application*.

Modernizing a complete application is more complex and takes longer than just modernizing it's navigation.

The example chosen is taken from a simple personnel management system:

Application before Modernization - 2.5 minutes Modernized Application - 2 minutes

This tutorial has these steps: Identify your business objects - 1 minute Create a prototype of your application - 3 minutes Create a filter and snap it in - 4 minutes Make a plan of the 5250 screens you will need to use - 2.5 minutes Identify the relevant screens and fields to newlook - 4 minutes Define the screens to the VLF and build a navigation script (New Employee) -7.5 minutes Define the screens to the VLF and build a navigation script (Employee Details) - 5 minutes Link the Selected Employee in the Instance List with the Display Employee Screen - 4 minutes Make Function Keys Go Somewhere Different - 4.5 minutes Handle Unexpected Stops in Navigation and Messages - 3 minutes Update the Instance List from 5250 Screens - 4 minutes

This tutorial is presented as a series of movies showing each step in the RAMP process. It is recommended that at the end of each movie you complete the outlined steps. This will reinforce the concepts and provide hands-on experience with RAMP. Some individuals may find it convenient to use a second machine or an extended desktop to view the movie while completing the tutorial.

Application before Modernization - 2.5 minutes

Play Movie to review the application before it was modernized or read the Movie Summary.

Movie Summary

For movie Application before Modernization - 2.5 minutes.

This movie shows a simple 5250 application for maintaining the details of employees which we are going to modernize.

It also introduces the concepts of Junction Screen and Destination Screen.

B Session A - [24 x 80]	
Browse/Maintain Employee and Skill FilesEmployee Number: : : : : : : : : : : : : : : : : : :	2000
Date Skl Skill Skill Acquired Code Description Comment 1/12/86 COM COMMUNICATIONS DEG 2/12/86 MANAGE2 MANAGEMENT COURSE 2 0/00/00 0/00/00	Grade P M +
F1=Help F3=Exit F4=Prompt F12=Cancel F14=Msgs F21=Change	
MA a 🗙 SYSTEM 👔	01/001

Modernized Application - 2 minutes

Play Movie to review the final output of this tutorial or read the Movie Summary.

Movie Summary

For movie Modernized Application - 2 minutes.

This movie shows how the modernized application we will create in the tutorial works:

🖇 Personnel				
File Edit View Actions Tools Hel	p (Framework) (Administration)			
🔇 New 🏾 🎽 Sign Off	🔲 Details 🛛 🥙 Transfer			
_Employees				
On Tool Bar	Personnel	_Employee	Description	<u>^</u>
🖃 🎒 Personnel	Employee Surname	A0193	SMITHSON FRED	
Employees	SM	A1001 A1002	SMITH ANDREW SMYTHE JOHN	_
	Employee Given Name(s)	A1002 A1003	SMITHE BOB	E
		A1004	SMITHSON PAUL	
	L	A1005	SMITHS PETER S	
		A1006	SMITHERS JACK	
		A1619 A2153	SMITH JOHN BLAXLAND	×
	Search		500118-1100	Clear List
	👹 _Employee : Details (A1004-SMITHSON PAUL)			
	Details Documents 💱 Timesheets Transfer			
Browse/Maintain Employee and Skill Files Save				
	Employee Number	A1004		
	Employee Surname	SMITHSON		Prompt
	Employee Given Name(s)	PAUL		
	Street No and Name Suburb or Town			
State and Country		NSW.	21	144
	Home Phone Number	419 5656		
	Department Code Section Code			
Start Date (DDMMYY)		1/05/80	·)	
	Termination Date (DDMMYY)		v)	
	Date Ski Skill Skill			
	Acquired Code Description	Co	omment Gr	ade
1/12/82 ₩ AC ACCOUNTANCY DE 1/02/83 ₩ MANAGE1 MANAGEMENT CO		GREE		Ρ
		RSE 1		Ρ
	1/02/84 MANAGE2 MANAGEMENT COU	IRSE 2		M
		Messages R	eady LANSA02 ENG	DCXUSER 28/03/06 9:56

Identify your business objects - 1 minute

Play Movie to learn how to identify your business objects or read the Movie Summary.

Movie Summary

For movie Identify your business objects - 1 minute

This movie shows how to work out what the business objects are for an application:

- See what words the end users use to describe what the system works with. These words are often reflected in the application menus and screen titles.
- In our sample application the users work with the details of Employees, so we decide we will create an Employees business object.

Create a prototype of your application - 3 minutes

Play Movie to learn how to create a prototype of your application or read the Movie Summary.

Movie Summary

For movie Create a prototype of your application - 3 minutes. This movie shows how to create a prototype of the modernized application:

- Start the Instant Prototyping Assistant
- Create business object _*Employees*
- Create two actions *New* and *Details*
- Associated the actions with _Employees
- Create a subsystem called *Personnel* and put the _Employees business object in it

See Also: Start the Instant Prototyping Assistant

Create a filter and snap it in - 4 minutes

Play Movie to learn how to create a filter and snap it in or read the Movie Summary.

Movie Summary

For movie Create a filter and snap it in - 4 minutes

This movie shows how to create a filter that locates employees:

Create code for filter

- Start the Program Coding Assistant
- Choose Filter that searches by all logical views
- Enter the name of the physical file that most resembles the business object: *PSLMST*
- The code assistant generates the code for the filter program
- Use the Copy to Clipboard button

Create filter reusable part

- Open the Visual LANSA development environment and:
- Create a new reusable part, call it by any unique name and give it a description
- Paste the generated code from the clipboard into the Source tab for the reusable part
- Compile it

Snap in the filter

- Return to the Visual LANSA Framework
- Close the Coding Assistant window
- Bring up the properties of the _Employees business object
- Go to the Filters tab and select the filter for this business object and snap in a real filter choosing the reusable part we just created
- Close the properties window, and save the Framework

You now have a real working filter. If you enter a value, all the matching employees will be loaded into the instance list.

See Also: Start the Program Coding Assistant

Make a plan of the 5250 screens you will need to use - 2.5 minutes

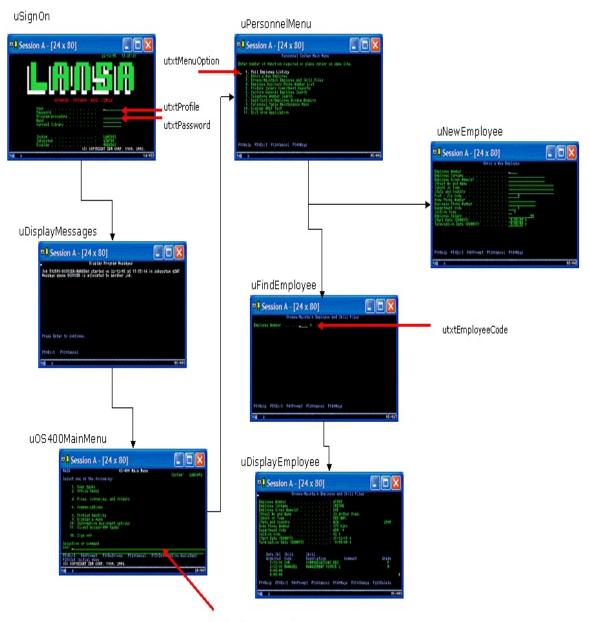
Play Movie to learn how to make a plan of the 5250 screens you will need to use or read the Movie Summary.

Movie Summary

For movie Make a plan of the 5250 screens you will need to use - 2.5 minutes. This movie shows how to make a plan of the screens that will be used:

- Draw a diagram of all the screens that you want to use
- Create a naming standard for screens and keep the names less than 32 characters.
- Assign a unique name to all the screens to be used as the form name in newlook (newlook must be able to differentiate all these screens, even though some have the same title)
- Decide which fields have data added to them as part of the navigation process and give these fields a name that is unique within the screen.

You now have a diagram that sets out what you have to define in newlook:



utxtSelectionOrCommand

Identify the relevant screens and fields to newlook - 4 minutes

Play Movie to learn how to identify the relevant screens and fields to newlook, or read the Movie Summary.

For movie Identify the relevant screens and fields to newlook - 4 minutes.

This movie shows how to identify the screens and fields in newlook. Use the diagram created in the previous step for naming (see Movie Summary):

Start newlook Client

• And then do the following for every screen:

Identify the screen

- Choose the Identify option in the Tools menu
- Check that the newlook screen identification area adequately identifies the screen (If the current recognition area is not sufficient to distinguish this screen, select a larger area, and use the pop-up menu to mark it as Screen Id.
- Save using the form name

Name the screen

- Choose the Designer option in the Tools menu
- Name the screen (form). The easiest way to do this is to double-click on an unused area of the screen, and edit the Name property of the form object. Note that names are case-sensitive, can be maximum 256 characters and do not allow trailing blank spaces.
- Right-click all fields on the form that are used for navigation (if any) to display their properties.
- Use the name property of the fields to identify them. For example on the Sign-on screen name the User field *utxtProfile* and the Password field *utxtPassword*.

Use the diagram in this Movie Summary to see which screens and which fields on the screens you need to identify.

Repeat these steps for each screen.

Define the screens to the VLF and build a navigation script (New Employee) - 7.5 minutes

Play Movie to learn how to define the screens to the VLF and build a navigation script (New Employee) or read the Movie Summary.

For movie Define the screens to the VLF and build a navigation script (New Employee) - 7.5 minutes.

This movie shows you how to define screens and track navigation to the New Employee screen.

Track navigation to New Employee screen

- Start RAMP Tools and start newlook.
- Connect to newlook server and sign on
- Display the Personnel Menu by typing this command on the command line on the IBM i Main menu screen:

lansa run pslsys partition(dem)

- Select the New Employee option
- Once in the New Employee screen cancel back to Sign Off so that the Framework can track the navigation

Remember to choose menu options by typing in the number and pressing Enter (if you click on a menu option with the keystroke tracking can't follow this).

Define screens

- The sign-on screen and all menu screens as Junction Screens
- The uDisplayMessages as a Special Screen
- The uNewEmployee as a Destination Screen

Generate scripts

• For every screen, use the pop-up menu options to generate scripts based on the tracking information

Specify the command tab where the screen is displayed

- Select the uNewEmployee destination screen and locate and check *Employees New* on the list of commands for all the business objects on the right-hand panel
- Save the Framework using the Save Framework button

Close and restart the Framework

• You will see the New Employee screen snapped in the Framework

Also See Screen Tracking Area Destination Screen Details

Define the screens to the VLF and build a navigation script (Employee Details) - 5 minutes

Play Movie to learn how to define the screens to RAMP and build a navigation script to the Display Employee screen, or read the Movie Summary.

For movie Define the screens to the VLF and build a navigation script (Employee Details) - 5 minutes.

This movie shows you to define screens and track navigation to the Display Employee screen.

Track navigation to Employee Details screen

- Start the RAMP tools and connect to newlook
- On the Personnel Menu type 3 to select Display Employee option
- Enter the identifier of any employee on the Find Employee Screen
- The destination screen uDisplayEmployee is displayed
- Go back along the pathway of screens so the Framework can track the navigation
- Notice F12 has taken us back from UDisplayEmployee directly to uPersonnelMenu, bypassing Find Employee. We need to go to Find Employee and use F12 from there to show the Framework how to get back from uFindEmployee

Define screens

- uFindEmployee as a Junction
- uDisplayEmployee as a Destination

Generate scripts

• Use the pop-up menu to automatically generate scripts based on the tracking information

Specify the command tab where the screen is displayed

• Associate the Employee Details screen with the Details command for the Employees business object.

Start the screen in edit mode

After closing and restarting, the Employee Details screen is working. However, to change an employee's details you have to first click on the Change button. In Windows screens are usually shown in edit mode. To do this:

• Start RAMP Tools

- Locate the Invoke script for uDisplayEmployee destination
- Find the last line in the script, and add an instruction to automatically press F21. That will put it into edit mode:

/* Send the key required to navigate to uDisplayEmployee

SENDKEY(KeyEnter);

/* Send the key required to navigate to put the screen into edit mode

SENDKEY(KeyF21);

•••

Now, when the user clicks on an employee, they can edit the details straight away.

Also see: Invoke Script

Link the Selected Employee in the Instance List with the Display Employee Screen - 4 minutes

Play Movie to learn how to link the selected employee in the instance list with the Display Employee screen, or read the Movie Summary

For movie Link the Selected Employee in the Instance List with the Display Employee Screen - 4 minutes.

This movie shows how to display the details of the employee selected in the instance list.

If we select an employee and the Details command, the invoke script of the screen executes and RAMP navigates through a number of junction screens and then shows us the uDisplayEmployee screen.

However, regardless of which employee we click on in the instance list, the details one and the same employee are shown. This is because the invoke script contains the hard coded number of the employee we chose when tracking navigation.

Change the Invoke script to display details for current employee

To change the script so it shows the details of the currently selected employee:

- Start RAMP tools
- Locate the invoke script for the uDisplayEmployee destination screen

Here is the line where the hard coded value for the employee identifier is entered into the Employee Code field:

SETVALUE("utxtEmployeeCode", "A1004");

To substitute the employee identifier with the currently selected entry:

- Select "A1004" including the quotes
- Right-click and choose *Current Instance List entry*
- Select *Alpha Key 1* from the submenu

This replaces the hard coded value with a special value that will contain the identifier of the employee that the user has selected:

SETVALUE("utxtEmployeeCode", objListManager.AKey1[0]);

Save the changes to the script. The Display Employee screen now shows the details of the employee selected in the instance list.

Also See Replacing Hardcoded Employee Number with Current Instance List Entry Invoke Script SETVALUE Function

Make Function Keys Go Somewhere Different - 4.5 minutes

Play Movie to learn how to make function keys go somewhere different or read the Movie Summary.

For movie Make Function Keys Go Somewhere Different - 4.5 minutes.

This movie shows how to automatically redisplay the Display Employee screen after the user has made a change and pressed Enter. It also shows how to hide function keys and buttons which are not required.

Automatically redisplay the screen

When the user edits an employee and presses Enter, they return to the Find Employee screen. We want to change this so that the Display Employee screen is redisplayed (this is how Windows typically works).

To redisplay the edit screen we locate the **Button Script** for the Display Employee destination and change it so that when the user presses Enter:

- First we tell RAMP to press Enter to go to the Find Employee screen
- Then we set the value of the employee code field on the Find Employee screen in the same way as in the previous tutorial
- And then press Enter to go to the Display Employees screen
- Lastly to start edit mode we add a script instruction to press F21

This is the code:

```
...
Case KeyEnter:
SENDKEY(KeyEnter);
SETVALUE("utxtEmployeeCode", objListManager.AKey1[0]);
SENDKEY(KeyEnter);
SENDKEY(KeyF21);
```

•••

Hide function keys and buttons

We also hide most of the buttons that are displayed on Display Employee because they are not required:

- Display the Destination Screen Details for Display Employee
- Disable and hide all buttons except Enter
- Change the caption of the Enter button to Save

Now save your changes and restart the Framework.

If you now change some employee details to valid values and press Save, the Display Employee screen is redisplayed.

Handle Unexpected Stops in Navigation and Messages - 3 minutes

Play Movie to learn how to handle unexpected stops in navigation and messages or read the Movie Summary.

For movie Handle Unexpected Stops in Navigation and Messages - 3 minutes. This movie shows how to handle unexpected stops in the navigation and how to issue messages.

Handle unexpected stops

If the employee details entered by the user are invalid, the user stays on the Display Employee screen instead of going to the Find Employee screen, and consequently our script instructions for handling the Enter key will be wrong. We need to change the script so that if it detects that the user did not get to the Find Employee screen, it stops and takes no further action. To do this: Go to the Button Script for the uDisplayEmployee Destinations and locate the SENDKEY(EnterKey) statement which takes us to the Find Employee screen.

Add this line under the SENDKEY statement to check if we actually got to the Find Employee screen and to end the script with a return if we did not:

if (CURRENT_FORM() != "uFindEmployee") return;

Issue a message

If we get past this line, we can assume that the employee details were successfully saved and issue a message. To do this add this line after the if statement:

ALERT_MESSAGE("Employee ", objListManager.AKey1[0], " has been saved.");

Commit your script changes and then save and restart the Framework.

Now, if we attempt to save an employee with invalid data, the script stops and the Display Employee screen is redisplayed.

If we abandon changes for that employee, click on another employee and successfully save the details, we get the message we created and the script continues on around to the Display Employee screen again.

Update the Instance List from 5250 Screens - 4 minutes

Play Movie to learn how to update the instance list from 5250 screens or read the Movie Summary.

For movie Update the Instance List from 5250 Screens - 4 minutes.

See topic Updating the Instance List from RAMP screens which summarizes the information in this movie.

Modernizing Application Navigation

This tutorial introduces the key concepts and steps required to *modernize the navigation of an existing 5250 application*.

This is the simplest and most rapid way to modernize an application and must be distinguished from modernizing a complete application, which is a more complex task.

However, it should be noted that in many cases modernizing a complete application yields better modernization results because it introduces more powerful means of organizing and accessing information.

The examples used in this tutorial are from the I5/OS operating system. These I5/OS objects are used:

- Job Queues
- Output Queues
- IFS Folders
- System Jobs

These objects were chosen because all System i users are familiar with them. Note however that the concepts in this tutorial apply equally to commercial business objects like Products, Orders, Customers, Invoices, Policies, etc.

To review the final output of this tutorial see Modernized Navigation - 5 minutes.

This tutorial has these steps:

Create a Prototype of Your Application - 9.5 minutes Identify Your 5250 Entry Point Screens Using newlook - 13 minutes Script the Screens and Snap them in the Framework - 16 minutes

Modernized Navigation - 5 minutes

Play Movie to review the final output of this tutorial or read the Movie Summary.

For movie Modernized Navigation - 5 minutes.

This movie shows the modernized application we will create in the tutorial:

👹 iSeries Server					. H		
File Edit View Actions Tools Help			-14				
🎽 Sign Off 🛞 New 🔻 🔗	Emalí 🚖 Print 🖂 📋	Details 💌 🗧	🔁 Transfer 🛛 💋 Ca	lculator			
System Jobs							
On Tool Bar	🍓 System Job						
ISeries Server IFS Folders Job Queues Output Queues System Tobs	Type options, press Enter.	W lapsed time	Vork with Active Job 00:00:00	Active jobs	21/12/05 208	LANSA02 (09:49:07	OK
Demo Application Programming Techniques	2=Change 3=Hold 8=Work with spooled files	4=End	5=Work with 13=Disconnect	6=Release	7=Display message		
	Opt Subsystem/Job LA@JSM JSMJOB QJVACMDSRV LA2JSM JSMJOB QJVACMDSRV LA2PGMLIB LISTENER TP00000001 Parameters or command	User QSYS LAUSER LAUSER QSYS QOTHPRDOWN QOTHPRDOWN QSYS LA2PGMLIB LA2PGMLIB	Type CPU % SBS .(ASJ .(BCI .(BSS .(BSS .(BSS .(BSS .(BSS .(BCI .(SBS .(BCI .(BCI .(BCI .(BCH .() CMD-RUNJSM) PGM-QJVACMDSR)) CMD-RUNJSM) PGM-QJVACMDSR))) CMD-LANSA	DEQW TIMW	More	
			Messages	Ready	Local ENG	DCXUSER 1	2/21/05 9:51 🕥

Create a Prototype of Your Application - 9.5 minutes

Play Movie to learn how to prototype your application or read the Movie Summary.

For movie Create a Prototype of Your Application - 9.5 minutes. This movie shows how to create a prototype of the modernized application.

Create a prototype application

- Start the Instant Prototyping Assistant
- Create business objects *Job Queues*, *Output Queues*, *IFS Folders* and *System Jobs*
- Create action *Work With*
- Associate *Work With* with all the four new business objects
- Create application *iSeries Server* and add the four business objects to it

Remove Filters

The Program Coding Assistant automatically creates filters for new business objects. In this tutorial we do not need them.

To remove the filters perform these steps for all the four business objects:

- Display the properties of the business object
- On the Filters tab select and delete the filter
- On the Commands Enabled tab change the Work with command to be a *Business Object Command*
- On the Command Display tab change the Object Command Presentation option to *Use All Of The Window*
- Save and restart the Framework

Execute your application prototype. Optionally type in text and insert images to the prototype screens so you can explain your proposed design to others.

Identify Your 5250 Entry Point Screens Using newlook - 13 minutes

Play Movie to learn how identify your 5250 entry point screens or read the Movie Summary.

For movie Identify Your 5250 Entry Point Screens Using newlook - 13 minutes.

This movie shows how to identify 5250 screens and fields on them so that we can reference them in the Visual LANSA Framework.

In this tutorial you identify these screens and fields:

Screen	Field			
Sign On	User profile and password			
Attempt Recovery	Menu option			
Sign Off				
Display Messages				
I5/OS Main Menu	Command to execute			
WRKJOBQ				
WRKOUTQ				
WRKLNK				
WRKACTJOB				

Identify the screens

• Start newlook 8.0

Then for every 5250 screen:

- Navigate to the screen
- Select the Designer option in the Tools menu
- Double-click the screen and assign a name to it (using the Form Name property) and when required, also for the fields on the screen
- Close the Designer using the File menu
- Specify a name for the screen and save

• Proceed to the next screen

You can do the naming at any time in any order and you don't have to name all the forms and fields in a single session.

Also See Start newlook

Script the Screens and Snap them in the Framework - 16 minutes

Play Movie to learn how to script screen navigation and snap screens in the Framework or read the Movie Summary.

For movie Script the Screens and Snap them in the Framework - 16 minutes

This movie shows how to script the 5250 screens and snap them to the Framework.

• (It first shows a quick tour of the RAMP Window)

Track Navigation

- Start newlook in the RAMP window
- Connect to the System i server
- Demonstrate to RAMP the navigation from the I5/OS main menu to WRKLNK, WRKJOBQ, WRKOUTQ and WRKACTJOB and back
- Sign off from the System i server

Define screens

- For every screen, click on the message that says the screen is unknown
- Define the Sign-on screen and the Main menu as Junction Screens
- Define WRKLNK, WRKJOBQ, WRKOUTQ and WRKACTJOB as Destination Screens
- Define Display Messages as a Special Screen

Create scripts

• Use the pop-up menu to automatically create scripts for all the screens

Enable Function Keys

- Select the WRKLNK screen in the Screen and Script List
- In the Destination Screen Details check *Page Up* and *Page Down* in the Enable for NL column in the Function Key Enablement Group so that the user can use these keys from the keyboard.

• Repeat this step for the other three destination screens

Specify the command tab where the screen is displayed

• In the *Associated Command Handlers* list associate the screens with the command tab:

WRKJOBQ	Job Queues - Work With	
WRKOUTQ	Output Queues - Work With	
WRKLINK	IFS Folders - Work With	
WRKACTJOB	System Jobs - Work With	

Specify a different Session Option for every screen

To execute the destination screens in separate sessions select the *Session* options SESSION_A, SESSION_B, SESSION_C and SESSION_D for the four destination screens respectively (normally the default value *AUTO is used which means the Framework will automatically determine the correct session)

Note that in addition to selecting the checkbox in front of the associated command handler you also need to click its name so that it is highlighted when you specify the Session Option.

Save and Restart the Framework

Open the System i Server application and click on any of the four business objects to start a new 5250 session to the System i server and display the modernized 5250 screen.

Scripting Tutorials

General

Introduction to Scripts - 6.5 minutes Reading, Writing and Storing Values in Scripts - 4 minutes Debug and Diagnostics - 2.5 minutes

Subfile Handling

Not Using a Datagrid Control - 1 minute Using Subfile Accessor - 5 minutes Subfile Direct Access - 2 minutes

Scripting

RAMP manages the 5250 screens in the modernized application with scripts.

Learning

Using

Debugging

Learning

The movie Introduction to Scripts - 6.5 minutes shows you how to learn scripting basics.

Types of Scripts in RAMP introduces you to RAMP scripts.

You also need to know how to Generate Scripts Automatically because this is how most scripts are created

The movie Reading, Writing and Storing Values in Scripts - 4 minutes shows how to pass values to and from your screens.

Javascript Essentials teaches you some basic techniques you will often use when writing scripts.

Introduction to Scripts - 6.5 minutes

•

Play Movie to learn scripting basics or read the Movie Summary.

For movie Introduction to Scripts - 6.5 minutes

Scripts are usually created automatically in RAMP, but to deal with them confidently you need to understand some scripting basics.

This movie introduces Javascript, the types of scripts in RAMP and tracing:

Basic Javascript syntax

- Comments are marked with /* */
- Lines are ended with a semicolon (;)
- Literals are enclosed in double-quotes (")
- There are Framework Objects that Scripts Can Refer To
- The structure of the conditional switch statement is:

```
switch(n)
{
  case 1:
    execute code block 1
    break
  case 2:
    execute code block 2
    break
  default:
    code to be executed if n is
    different from case 1 and 2
}
```

Types of scripts in RAMP

The movies shows:

- The interaction between Invoke Scripts and Return Scripts
- Button Script
- Eliminate Script

Application level tracing

• Use the Tracing option in the Framework menu to start Application Level

Tracing

Types of Scripts in RAMP

There are different types of RAMP scripts for different types of screens:

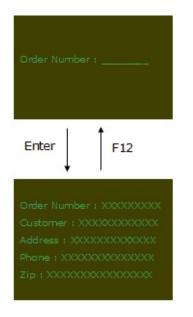
Destination Screens	 An Invoke Script indicates how the destination screen should be invoked (or accessed). This script is the key to how the Framework manages screen navigation. A Return Script indicates how to cancel the screen and get back to the nearest junction screen. A Button Script indicates how buttons clicked by the user should be handled.
Junction Screens	A Navigate Script indicates how to navigate from one screen to another
Special Screens	An Eliminate Script indicates how to hide them.

For a basic scripting example, also see Scripts in a Classic Details Display.

Scripts in a Classic Details Display

This example shows how two very simple 5250 screens are modernized in RAMP.

The first screen GETORDER asks for an order number to be input and the second screen SHOWORDER displays the order details:



The user repeats order inquiries by using the F12 function key.

Modernized Version

To modernize the application we identify the 5250 screens and script their interaction to RAMP:

- The GETORDER screen becomes a junction screen. It will not be displayed.
- The SHOWORDER screen becomes a destination screen.

orders in many ways.	Se COLOR	R0001 Order number 1 R0002 Order number 2 R0003 Order number 3 R0004 Order number 4 R0005 Order number 5 R0007 Order number 5 R0007 Order number 6 R0007 Order number 6 R0007 Order number 7 R0009 Order number 8 R0009 Order number 9 Mohtmann Order number 9	
Grow : Detais (DIDE 5250 Display Order			
This panel will display list above. It will have	the \$250	ted from the instance	° ⁴ Order Number :
an order.	Customer : XXXXXX Address : XXXXXXX		
	Phone : XXXXXXXXX	~~~~~	/

SHOWORDER DESTINATION 5250 SCREEN

In the modernized application the user selects orders from the instance list and the SHOWORDER screen shows the details of the selected order.

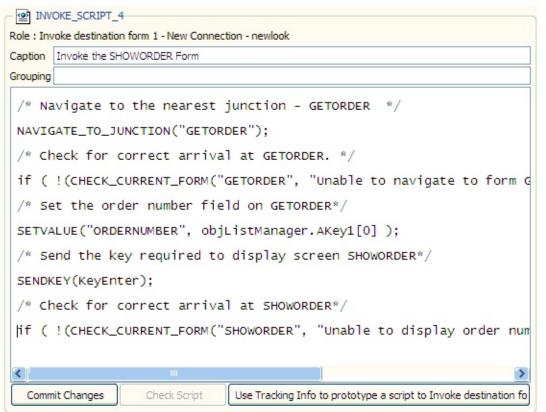
SHOWORDER has three scripts:

- The Invoke Script which displays the SHOWORDER screen.
- A Return Script which contains a single executable line to cancel out of the SHOWORDER screen.
- A Button Script which will not be executed for SHOWORDER because all function keys are hidden and disabled in this screen (the user just clicks on different orders up in the instance list to display the details of a different order.)

Every time a user clicks on an order in the instance list, the SHOWORDER's return script is executed to return to the navigation network. Once there, the invoke script is executed to display the SHOWORDERS screen with the details of the selected order.

Invoke Script

Every Destination Screen has an invoke script which controls how it is displayed. Here is an example of a script that invokes a SHOWORDER screen when an order is selected in the instance list:



First the script navigates to the 5250 junction screen GETORDER which is used to select which order is to be shown:

NAVIGATE_TO_JUNCTION("GETORDER");

It then makes sure that we get to the GETORDER screen. If this check fails an error message is shown:

if (!(CHECK_CURRENT_FORM("GETORDER", "Unable to navigate to form GETORDER"))) return;

Next the script retrieves the current order number from the instance list to the GETORDER screen. Typically you need to edit this part of the script (see Replacing Hardcoded Employee Number with Current Instance List Entry):

SETVALUE("ORDERNUMBER", objListManager.AKey1[0]);

And then presses the Enter key to process the GETORDER screen:

SENDKEY(KeyEnter);

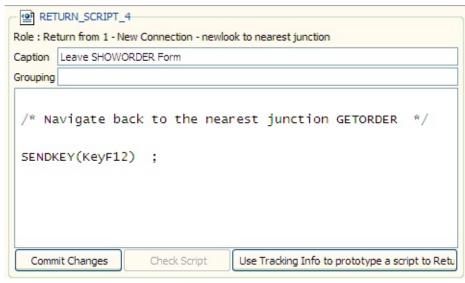
Finally the script makes sure that screen SHOWORDER has arrived back from the System i and is ready to be displayed. If this check fails, an error message is shown:

if (!(CHECK_CURRENT_FORM("SHOWORDER", "Unable to display order number " + objListManager.AKey1[0]))) return;

Also See: NAVIGATE_TO_JUNCTION Function CHECK_CURRENT_FORM Function SETVALUE Function SENDKEY Function

Return Script

Every **Destination Screen** has a return script which indicates how to exit the destination screen:



The return script shows how to go back to the nearest junction and onto the navigation system. It probably contains just a single executable line:

SENDKEY(KeyF12);

The return script does not necessarily have to go back via the junction where it was originally invoked from, but typically it does.

Also See:

SENDKEY Function

Button Script

Every Destination Screen has a button script which indicates how the function keys the user presses or the buttons the user clicks should be handled:

```
BUTTON_SCRIPT_4
Role : 1 - New Connection - newlook
Caption wrkjobg - Handle function keys and button usage
Grouping TUTORIAL 1
 /* Handle function keys and buttons for wrkjobg */
 switch (objScriptInstance.FunctionKeyUsed)
    case KeyEnter:
       SENDKEY(KeyEnter);
       break;
    case KeyF3:
       SENDKEY(KeyF3);
       break;
    case KeyF4:
        SENDKEY(KeyF4);
       break;
    case KeyF5:
        SENDKEY(KeyF5);
       break;
    case KeyF12:
       SENDKEY(KeyF12);
       break;
    case KeyF24:
        SENDKEY(KeyF24);
        break;
    default:
        SENDKEY(objScriptInstance.FunctionKeyUsed);
       break;
 }
  Commit Changes
                    Check Script
                                  Use Tracking Info to prototype a script to 1 - N
```

When you generate a button script from tracking information, every key present on the screen is handled in a series of SENDKEY functions in the script.

```
...
case KeyEnter:
SENDKEY(KeyEnter);
```

•••

If you disable a key in the Destination Screen Details the script for it does not get executed.

In most situations navigational 5250 function keys like F12=Cancel and F3=Exit should be disabled and not shown because they are not required in

Windows navigation and tend to just confuse users.

You can also add buttons or function keys which are entirely processed on the client system and never sent back to the server. To do this you enable them in the Destination Screen Details and then add handling for them in the button script.

Also See:

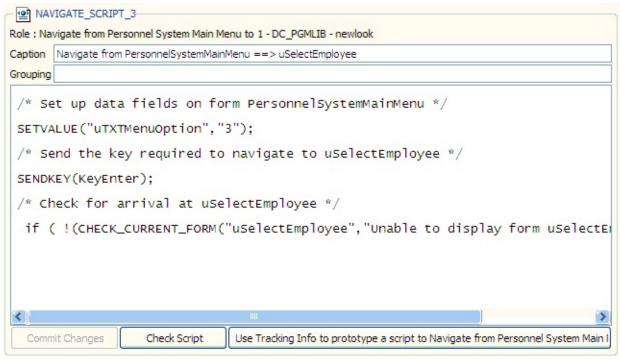
SENDKEY Function

Navigate Script

Junction Screens control application navigation but are never shown to the user.

These screens have navigate scripts associated with them which control the navigation to and from the junction screen.

This example script selects menu option 3 on a menu, emulates the pressing of the enter key and checks that the correct screen is displayed:



Typically there are one or more scripts navigating away from a junction screen towards other junctions and one or more scripts navigating towards it from other junctions.

Collectively these scripts define a navigation network between junctions.

Also See:

SETVALUE Function

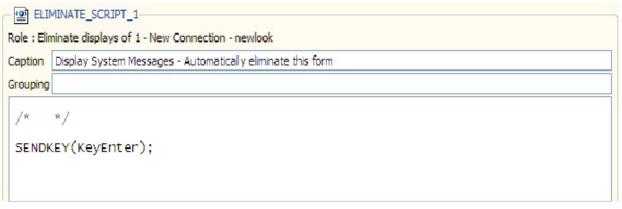
SENDKEY Function

CHECK_CURRENT_FORM Function

Eliminate Script

Special Screens have an eliminate script associated with them. These scripts define what is to happen whenever a special screen appears so as to eliminate it from the 5250 data stream and make it invisible to other scripts.

This script eliminates the display of the system messages screen:



Also See: SENDKEY Function

Generate Scripts Automatically

After you have tracked the navigation in your application using RAMP Tools, you can automatically generate scripts using the tracking information

To generate scripts, click on the messages in the Message Area and use the popup menu.

To see how it is done you can play the tutorial movie Define the screens to the VLF and build a navigation script (New Employee) - 7.5 minutes.

The best sequence for choreographing 5250 screen interactions is to:

- Track all the appropriate 5250 screen interactions in the navigation path.
- Go down through the tracking area and classify every screen as a destination, special, junction, etc.
- Go down through the tracking area again and for each screen ask for the scripts to be automatically generated.

Reading, Writing and Storing Values in Scripts - 4 minutes

•

Play Movie to learn how to read and write values in scripts and how to store them as variables or as properties of objects or read the Movie Summary.

Movie Summary

For movie Reading, Writing and Storing Values in Scripts - 4 minutes This movie shows how you can read, write and store values in scripts.

Reading values

```
Scripts can read values from the instance list like this:
myVariable = objListManager.Akey3[0];
```

See Visual and Programmatic Identifiers.

If the user has selected several entries in the instance list, you can read all the values in a loop like this:

```
var i = 0;
var strMessage = "";
for (i = 1; i <= objListManager.TotalSelected; i++)
{
    strMessage += "Selected Employee " + objListManager.AKey3[i] + " ";
}
alert(strMessage);
```

Or from a field defined on a 5250 screen like this: MyVariable = GETVALUE("utxtEmployeeCode");

Writing values

The script can put values on the screen like this: SETVALUE("utxtEmployeeCode", "myText");

Storing values

You can store values in Javascript variables and then read and write from them: Var MyString = ""; MyString = objListManager.Akey3[0];

These variables exist only while the script is running. To share information

between scripts, you need to create and set a property for objGlobal: objGlobal.uLastValue = "anything";

Then another script can read this value: myVariable = objGlobal.uLastValue;

Getting script pieces quickly

Using the Scripting Pop-up Menu

Javascript Essentials

RAMP manages the 5250 screens in the modernized application with JavaScript scripts.

JavaScript is the most commonly used scripting language in the world. You can also use Microsoft's JScript extension. Note that JavaScript skills can be used in many other contexts such as LANSA for the Web and HTML manipulation.

This section describes some Javascript essentials:

External JavaScript Documentation

Alert()

Converting Numbers to Strings

Converting String to Numbers

String Manipulation Functions

Is This Variable Number or String?

Using the objGlobal Object

External JavaScript Documentation

Put this link behind an icon on you desktop for instant access to formal JavaScript documentation:

http://www.w3schools.com/jsref/

There are also many good books available (such as JavaScript Bible by Danny Goodman, ISBN 0-7645-3188-3).

Alert()

The Alert() function is your most useful tool for debugging errant scripts.

For example:

Alert("About to send the enter key"); Alert("The value of x is " + x.toString()); Alert("The customer number is " + objGlobal.CustomerNumber);

Also See Strange behavior in scripts Object expected

Converting Numbers to Strings

If you have a number in JavaScript variable and you want to convert it to a string use the toString() function. For example:

```
var number = 5.65;
var stringnumber = number.toString();
```

alert(stringnumber);

SETVALUE("Amount",stringNumber); SETVALUE("Amount",number.toString());

Converting String to Numbers

If you have a string and want to convert it to a number then use the parseInt() method. For example this script returns integer values containing 1234 and 43 respectively into X:

X = parseInt("1234",10);

X = parseInt("34abc",10);

The second argument (10) specifies you want to use a base 10 numbering system. It's unusual to use anything for this parameter except 10 and you should always specify it as the default is a bit unpredictable. (See, for example, http://www.w3schools.com/jsref/jsref_obj_global.asp if you are interested as to why)

If you need to have decimals then use parseFloat(). For example this script returns floating point values 1234.345 and 34.7 respectively into X:

X = parseFloat("1234.345"); X = parseFloat("34.7abc");

Remember that these are floating point values so they are not always as accurate or as predictable as signed or packed decimals numbers.

String Manipulation Functions

String variables in JavaScript have a number of very useful string functions. Here's a sample of the most commonly used:

Operation / Function	Example
Concatenation (+)	<pre>var S1 = "Customer"; var S2 = "123456"; var S3 = S1 + " " + S2 + "could not be found"; puts Customer 123456could not be found in variable S3.</pre>
IndexOf – finds first occurrence of a string in a string	<pre>/* 012345678901 */ var S1 = "ABCDHELLOABC"; var pos = S1.indexOf("HELLO"); will put the number 4 into variable pos.</pre>
lastIndexOf - finds last occurrence of a string in a string	/* 012345678901 */ var S1 = "ABCDHELLOABC"; var pos = S1.lastIndexOf("AB");
	will put the number 9 into variable pos.
charAt – returns the character at a specific position in a string	/* 012345678901 */ var S1 = "ABCDHELLOABC"; var S2 = S1.charAt(4); var S3 = S1.charAt(9);
	will put "H" into S2 and "A" into S3.
length – returns the length of a string	/* 012345678901 */ var S1 = "ABCDHELLOABC"; var I = S1.length;
	will put the number 11 into variable I.
substring – returns the	/* 01234567789 */

substring of string using a starting and ending point.	var a = "Hello World"; var b = a.substring(4,8);
	will put "o Wor" into b.
substr – returns the substring of a string using a starting position and a	/* 01234567789 */ var a = "Hello World"; var b = a.substr(2,3);
length	will put "llo" into b.
toLowerCase – returns the	var a = "Hello World";
lowercase of string	var b = a.toLowerCase(); will put "hello world" into b.
toUpperCase – returns the uppercase of a string	var a = "Hello World"; var b = a.toUpperCase();
	will put "HELL WORLD" into b.

There are more string functions like these available. See:

http://www.w3schools.com/jsref/jsref_obj_string.asp for more details.

Is This Variable Number or String?

Sometimes you have a variable in Javascript and do not know whether it is a number or a string. You can test the type of a variable by using the typeof() operator like this:

```
Var x = 1.234;
Var y = "Hello";
Var Type1 = typeof(x);
Var Type2 = typeof(y);
```

```
Alert(Type1 + " and " + Type2);
```

This code displays the message "number and string".

There are six possible values that typeof returns: "number," "string," "boolean," "object," "function," and "undefined." The most useful are "number", "string" and "undefined".

"undefined" is useful because it tells you that something does not exist yet (ie: it's undefined) so sometimes you see code like this:

```
if (typeof(objGlobal.CustomerNumber) = "undefined"))
objGlobal.CustomerNumber = "12345";
```

Using the objGlobal Object

objGlobal is one of the Framework objects that scripts can refer to. Its purpose is to store your own properties.

This section shows some techniques in using it:

Getting Organized

Using objGlobal to pass optional parameters

Using objGlobal to pass optional parameters to an INVOKE script

Using objGlobal to define commonly used functions

Getting Organized

If you make a lot of use of the objGlobal object then you should look to organizing its use in some way. One way is to divide it up into multiple sub-objects by application or usage.

For example, if you did this in you logon script:

objGlobal.AppA = new Object(); objGlobal.AppB = new Object(); objGlobal.AppC = new Object();

Then in your scripts you could make sure your references do not accidentally interfere with each other.

For example objGlobal.AppA.CurrentCustomer is a different variable to objGlobal.AppB.CurrentCustomer and objGlobal.AppC.CurrentCustomer.

Using objGlobal to pass optional parameters

Extending the idea in the previous section slightly, you can introduce the concept of optional parameters being passed into scripts. In a script that needs to pass some optional parameters into another script you might find code like this:

```
objGlobal.OptParms = new Object();
objGlobal.OptParms.CustNumber = "12345";
objGlobal.OptParms.CustName = "ACME ENGINEERING";
NAVIGATE_TO_DESTINATION("uShowCustomer");
```

and the script that receives the optional parameters you would find code possibly structured something like this:

```
var CustNumber = "some default value";
var CustName = "some default value";
```

```
if (objGlobal.OptParms != null)
```

```
{
  CustNumber = objGlobal.OptParms.CustNumber;
  CustName = objGlobal.OptParms.CustName;
  objGlobal.OptParms = null;
}
```

/* Now we proceed to use the values in CustNumber and CustName */

The line **objGlobal.OptParms = null;** line is very important to this style of processing because it destroys the temporary OptParms object.

Using objGlobal to pass optional parameters to an INVOKE script

Sometime an INVOKE script is executed in different ways. For example:

- When the user clicks on a line in instance list, the script is invoked to display the customer details.
- This script may also be invoked from another script to display the details of a specific customer.

Since this script can be used in two different ways, it needs to be aware of what it is being asked to do. The easiest way to do this is to use the "ObjGlobal" object to pass *optional parameters* to it.

In an INVOKE script you can define and check for the existence of optional parameters like this:

/* Conceptually this script's behavior is controlled by 2 parameters which may or may not be passed to it */

var Parameter1 = "parameter default value"; var Parameter2 = "parameter default value";

/* If either parameter has been passed in the objGlobal object then override the default behavior. */

/* Note the destruction of the optional parameters. This is so they do not hang around to interfere $\ */$

/* with later executions of this script. They are created, passed into the script and then destroyed. */

if (objGlobal.optParameter1 != null) { Parameter1 =
objGlobal.optParameter1; objGlobal.optParameter1 = null; }

if (objGlobal.optParameter2 != null) { Parameter2 =
objGlobal.optParameter2; objGlobal.optParameter2 = null; }

/* Now use the values in Parameter1 and Parameter 2 to control how this script behaves */

< etc > < etc > As a specific example, imagine an INVOKE script that by default displayed the current customer from the instance list. However, some other scripts reuse it to display a specific customer, which may or may not be in the instance list. You could handle this situation like this:

/* By default this script displays the current customer from the instance list, so get the customer number */

```
var RequestedCustomer = objListManager.AKey1[0];
```

/* If the caller has supplied a specific customer number use it instead (making sure to destroy the optional parameter) */

```
if (objGlobal.optRequestedCustomer != null)
{
    RequestedCustomer = objGlobal.optRequestedCustomer;
    objGlobal.optRequestedCustomer = null;
}
```

/* Now display the details of the customer identified in RequestedCustomer */

< etc > < etc >

In a script that wants to display a specific customer number you could do something like this:

```
/* Save the changes and (re)display this customer */
case KeyEnter:
{
    var CustomerNumber = GETVALUE("CustNo"); /* Get the
```

Using objGlobal to define commonly used functions

If you want to create a JavaScript function that is reused in many places you could do something like this in your sign-on script:

```
objGlobal.Mult = function (x,y) {
    var z = x * y;
    return(z); }
objGlobal.Add = function (x,y) {
    var z = x + y;
    return(z); }
```

These operations define 2 functions in objGlobal named Mult and Add and the code that they contain.

Once this has been done the functions objGlobal.Add and objGlobal.Mult can be executed in other scripts like this:

```
var q = objGlobal.Add(222,3);
alert( q.toString() );
q = objGlobal.Mult(22,33);
alert( q.toString() );
```

which would display the results 225 and 726 respectively.

Using

Interacting with Instance Lists in Scripts Using the Scripting Pop-up Menu Updating the Instance List from RAMP screens Subfiles/Browselists Handling Pop-Ups Script Functions Framework Objects that Scripts Can Refer To User-defined script functions Switching Off Recursion Checking

Interacting with Instance Lists in Scripts

For an introduction to this topic, play the tutorial movie Link the Selected Employee in the Instance List with the Display Employee Screen - 4 minutes.

The instance list is the list of business object instances typically displayed in the upper right corner of the Framework window. For example, the shipped demonstration system uses an Employee business object that has an instance list that looks like this (outlined in red):

🔕 by Name 🔗 by Location 📓 by Skill 🔍 Other	Number	Name	Phone	Address 🔺
Specify a full or partial employee name.		BEN JONES	799 5268	144 Froc
		PATRICK PAUL	687 1717	6 Camillo
Employee Surname	A1013	GEORGE PATTISON	750 2562	12 Augu
	A1015	BRADLEY WOODS	450 1236	59 Darle
	A 1020	ADAM PETER DOUGLAS	674 5310	6 Readin
	A1021	DAVID MCCULLY	762 1321	15 Baker
Clear the current list of employees Search	A1025	MARY ROBINSON	126 3598	14 Whitt 🔽
	<			>

Many scripts need to interact with the instance list. These topics explain how to do it:

The List Manager Visual and Programmatic Identifiers Working with All Selected Entries

The List Manager

Script interactions with an instance list are done by accessing properties of the Framework JavaScript object named objListManager (the list manager).

For example an invoke script that displays a screen showing the details of an employee uses the objListManager in the SETVALUE command to set the employee to the selected entry in the instance list:

/* Navigate to the nearest access junction */
NAVIGATE_TO_JUNCTION("uFindEmployee");

/* Check for arrival at uFindEmployee */
if (!(CHECK_CURRENT_FORM("uFindEmployee","Unable to navigate to
form uFindEmployee"))) return;

/* Set the employee to be displayed to the employee selected in the */
/* instance list (which is identified by the programmatic identifier AKey3) */
SETVALUE("utxtEmployeeCode",objListManager.AKey3[0]);

/* Send the key required to navigate to uDisplayEmployee */
SENDKEY(KeyEnter);
SENDKEY(KeyF21);

Also See objListManager Replacing Hardcoded Employee Number with Current Instance List Entry

Visual and Programmatic Identifiers

Instance list entries always have an *identification protocol* that defines their visual and programmatic identification. You set these identifiers when you create the filter that controls the instance list.

(Refer to the section List Manager in the Framework guide if you want detailed information about the identification protocol.)

For example this LANSA command in a filter for employees adds entries to the instance list and sets programmatic and visual identifiers and additional columns for them:

Invoke Method(#avListManager.AddtoList) Visualid1(#Empno) Visualid2(#FullName) Akey1(#Deptment) Akey2(#Section) Akey3(#Empno) AColumn1(#PhoneHme) AColumn2(#Address1) nColumn1(#PostCode)

In this identification protocol:

- The third programmatic identifier (called AKey3) contains the employee number.
- The second visual identifier (called VisualId2) contains the employee's name.

When you know the identification protocol, you can create a JavaScript that displays the number and name of the currently selected employee in the instance list:

```
/* Get the current instance list details */
```

```
{
```

```
var strEMPNO = objListManager.AKey3[0]; /* 3rd Akey is the number */
var strNAME = objListManager.VisualId2[0]; /* 2nd VisualId is the name */
alert("Current employee number is " + strEMPNO);
alert("Current employee name is " + strNAME);
}
```

Like this:

Microso	ft Internet Explorer 🛛 🛛 🛛	
⚠	Current employee number is A 1020	
	ОК	
Microso	ft Internet Explorer	
Microso	ft Internet Explorer Current employee name is ADAM PET	ER DOUGLAS

Working with All Selected Entries

More than one entry can be selected in the instance list. This script displays the number and name of all selected employees in a message:

```
/* Get all the selected employees */
{
  var i = 0;
  var strMessage = "";
  for (i = 1; i <= objListManager.TotalSelected; i++)
  {
    strMessage += "Employee " + objListManager.AKey3[i];
    strMessage += " - " + objListManager.VisualId2[i] + "\x0D";
  }
  alert(strMessage);
}</pre>
```

So if this script was used with three selected instance list entries like this:

Number	Name	Phone	Address 木
A1001	BEN JONES	799 5268	144 Froc
A1012	PATRICK PAUL	687 1717	6 Camillo
A1013	GEORGE PATTISON	750 2562	12 Augu
A1015	BRADLEY WOODS	450 1236	59 Darle
A1020	ADAM PETER DOUGLAS	674 5310	6 Readin
A1021	DAVID MCCULLY	762 1321	15 Baker
A1025	MARY ROBINSON	126 3598	14 Whitt 🔽
<			>

It would display this alert message:

Microsoft Internet Explorer		
<u>.</u>	Employee A1015 - BRADLEY WOODS Employee A1020 - ADAM PETER DOUGLAS Employee A1021 - DAVID MCCULLY	
	ОК	

Using the Scripting Pop-up Menu

You can use the scripting pop-up menu to format and edit your scripts. To display the menu, right-click the Script Details area.

Cu <u>t</u>	
Copy	
Paste	
<u>U</u> ndo	
<u>R</u> edo	
Upper Case	
Lower Case	
Larger font	
Smaller font	
Show Line Numbers	
Current Framework	۲
Current Application	۲
Current Business Object	۲
Current Command	۲
Current Instance List Entry	۲
5250 Subfile Handling	۲
Session Control	۲

The first set of options Cut, Copy, Paste, Undo and Redo are commonly used options in many editors and are self-explanatory.

The Upper Case and Lower Case options will change the case of any text currently selected in the script editor. Note that Javascript is case-sensitive.

The Lower font and Larger font options allow you to change the size of the font being used by the text editor.

The Show Line Numbers option displays (or hides) line numbers in the text editor.

Use The Current... options to insert properties for various Framework objects into your script. Use:

- Current Framework to enter properties of objFramework
- Current Application to enter properties of objApplication
- Current Business Object to enter properties of objBusinessObject
- Current Command to enter properties of objCommand
- Current Instance List Entry to enter properties of objListManager

Use the 5250 Subfile Handling options to insert code for Subfiles/Browselists.

Use the Session Control options to enter commonly used functions and objUser parameters to your script.

Examples:

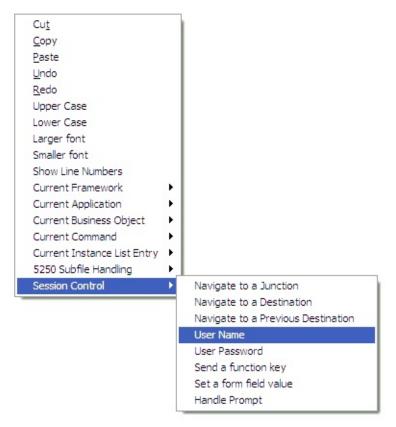
- Replacing Hardcoded User Name with Current Framework User
- Replacing Hardcoded Employee Number with Current Instance List Entry
- Adding Your Own Options to the Scripting Pop-Up Menu

Replacing Hardcoded User Name with Current Framework User

To replace the hardcoded user name "QPGMR" in this line of script with the name of the current framework user:

SETVALUE("utxtUserName", "QPGMR");

Select "QPGMR" (including the quotes), right-click and select the Session Control and then User Name option:



The constant "QPGMR" is now replaced with the substitution value for the current Framework user:

SETVALUE("utxtUserName", objUser.Name);

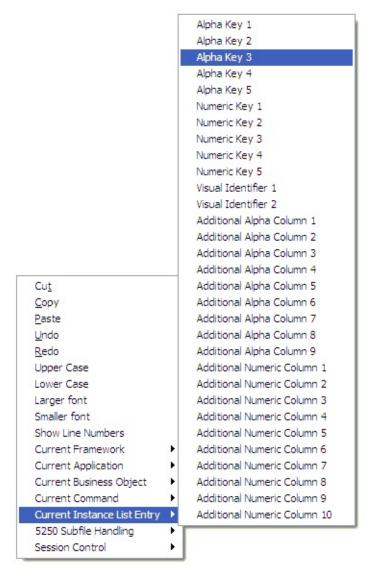
Replacing Hardcoded Employee Number with Current Instance List Entry

When you automatically generate scripts using tracking information, the scripts will contain the hardcoded field values you typed. To make the script to work with any selected object, you need to replace the hardcoded value with the appropriate identifier.

To replace the hardcoded employee number "A1234" in this line of script with the name of the employee currently selected in the instance list:

SETVALUE("uEmpNo","A1234");

First find out the Visual and Programmatic Identifiers used to identify the employee. Then highlight the hardcoded number "A1234" (including the quotes) in the script, right-click to bring up the pop-up menu, select the Current Instance List Entry option and select the appropriate identifier:



The constant "A1234" is now replaced with the programmatic identifier of the employee number:

SETVALUE("uEmpNo", objListManager.AKey3[0]);

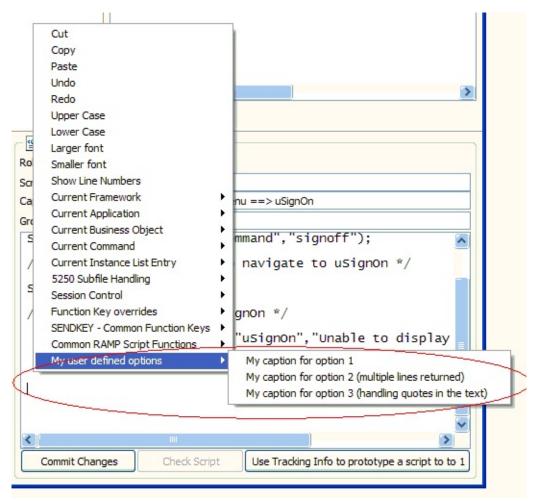
Adding Your Own Options to the Scripting Pop-Up Menu

You can add your own options to the scripting pop up menu by creating an xml file called uf_um835.xml, and putting it in the partition execute directory. You can do this using notepad.

This is an example of uf_um835.xml that you could create:

```
<?xml version="1.0"?>
<EXTRACT>
<MENUITEM>
<PROPERTY NAME="CAPTION" VALUE="My user defined options" />
<SUBMENUS>
<SUBMENUITEM>
<PROPERTY NAME="CAPTION" VALUE="My caption for option 1" />
<PROPERTY NAME="STRING" VALUE="My returned text for option 1" />
</SUBMENUITEM>
<SUBMENUITEM>
<PROPERTY NAME="CAPTION" VALUE="My caption for option 2
(multiple lines returned)" />
<PROPERTY NAME="STRING" VALUE="My returned line 1 for option 2" />
<PROPERTY NAME="STRING" VALUE="My returned line 2 for option 2" />
<PROPERTY NAME="STRING" VALUE="My returned line 3 for option 2" />
</SUBMENUITEM>
<SUBMENUITEM>
<PROPERTY NAME="CAPTION" VALUE="My caption for option 3
(handling quotes in the text)" />
<PROPERTY NAME="STRING" VALUE="Quotes and greater than and less
than need special handling" />
<PROPERTY NAME="STRING" VALUE="Quote - &quot;" />
<PROPERTY NAME="STRING" VALUE="Less than - &lt;" />
<PROPERTY NAME="STRING" VALUE="Greater than - &gt;" />
</SUBMENUITEM>
</SUBMENUS>
</MENUITEM>
</EXTRACT>
```

If you create a file called uf_um835.xml and paste this text into it and then put uf_um835.xml into your partition execute directory, you will be able to see these new options when you are editing RAMP scripts:



If you choose option 1, this will be added to your script: My returned text for option 1

If you choose option 2, this will be added to your script:

My returned line 1 for option 2 My returned line 2 for option 2

My returned line 3 for option 2

If you choose option 3, this will be added to your script: Quotes and greater than and less than need special handling Quote - " Less than - < Greater than - >

In the xml above, you can see that the caption displayed for the first submenu

comes from the caption property, and the value returned to the script when the user clicks on this submenu comes from the String property:

<SUBMENUITEM>

<PROPERTY NAME="CAPTION" VALUE="My caption for option 1" /> <PROPERTY NAME="STRING" VALUE="My returned text for option 1" /> </SUBMENUITEM>

From option 2, you can see how to return multiple lines when the user clicks on a submenu:

<SUBMENUITEM>

<PROPERTY NAME="CAPTION" VALUE="My caption for option 2 (multiple lines returned)" />

<PROPERTY NAME="STRING" VALUE="My returned line 1 for option 2" /> <PROPERTY NAME="STRING" VALUE="My returned line 2 for option 2" /> <PROPERTY NAME="STRING" VALUE="My returned line 3 for option 2" /> </SUBMENUITEM>

And from option 3, you can see the special handling if you want quotes (or greater than or less than) in the value returned to the script:

<SUBMENUITEM>

<PROPERTY NAME="CAPTION" VALUE="My caption for option 3 (handling quotes in the text)" /> <PROPERTY NAME="STRING" VALUE="Quotes and greater than and less than need special handling" /> <PROPERTY NAME="STRING" VALUE="Quote - "" /> <PROPERTY NAME="STRING" VALUE="Less than - <" /> <PROPERTY NAME="STRING" VALUE="Greater than - >" /> </SUBMENUITEM>

As long as your xml is valid xml, and keeps to the structure of the example above (EXTRACT, MENUITEM, SUBMENUS and SUBMENUITEM) it should work.

Note: Ensure that your version of UF_UM835.xml is backed up.

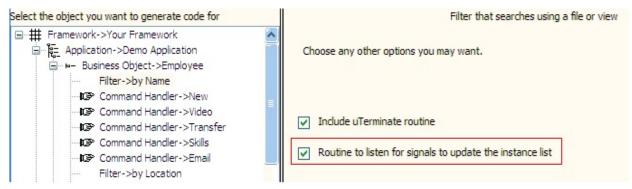
Updating the Instance List from RAMP screens

The tutorial movie Update the Instance List from 5250 Screens - 4 minutes covers this topic in detail.

A filter manages its associated instance list. When a RAMP screen deletes, adds or changes business object instances, it needs to notify the filter that a change has occurred.

Create the Filter with Program Coding Assistant

To create a filter that listens for changes from RAMP screens use the Program Coding Assistant and select the option *Routine to listen for changes and update the instance list:*



This option creates Filter Code which Automatically Handles Changes to Instance List.

Add AVSIGNALEVENT Function to the Button Script

Add an AVSIGNALEVENT Function in the button script of your RAMP destination screen for the button that handles the change (typically Save or Delete) to signal to the filter that the instance list needs to change.

For example, in a RAMP screen that updates an object, add this statement to its SAVE button script:

AVSIGNALEVENT("Update_List_Entry", "BUSINESSOBJECT", objListManager.AKey1[0]);

The event being signaled is named Update_List_Entry, and the value being passed is the identifier of the instance that has been updated.

To handle the saving of a newly created object, you must pass to the filter the identifier of the object. For example, to add a new employee with employee number, you would first capture the employee number on the screen using the GETVALUE Function and store it as a property of the objGlobal object, and then pass it to the filter:

```
objGlobal.utxtEmployeeCode = GETVALUE("utxtEmployeeCode");
SENDKEY(KeyEnter);
AVSIGNALEVENT("Add_List_Entry", "BUSINESSOBJECT",
objGlobal.utxtEmployeeCode);
```

(The utxtEmployeeCode field is the employee number field that has been defined as a text field on the destination screen.)

The standard event names you can use to update the instance list are:

- Refresh_Instance_List
- Update_List_Entry
- Add_List_Entry
- Delete_List_Entry.

Filter Code which Automatically Handles Changes to Instance List

This RDMLX code which is created by the Program Coding Assistant automatically handles events signaled by the RAMP screen (it is shown here just for your reference, you do not need to modify it):

```
* Handle any external requests to update the Instance List
* _____
EvtRoutine #Com_owner.avEvent WithId(#EventId) WithAInfo1(#AInfo1)
WithAInfo2(#AInfo2) WithAInfo3(#AInfo3) WithAInfo4(#AInfo4)
WithAInfo5(#AInfo5) WithNInfo1(#NInfo1) WithNInfo2(#NInfo2)
WithNInfo3(#NInfo3) WithNInfo4(#NInfo4) WithNInfo5(#NInfo5)
* put the received values into fields
Change #vf_elIdn #EventId.Value
* Map the AInfo and NInfo values passed, into the key fields - #EMPNO
Change #DEPTMENT #AInfo1
Change #SECTION #AInfo2
Change #EMPNO #AInfo3
Case #vf elIDN
when '= Refresh Instance List'
* Reload the Instance List
Invoke #Com_Owner.uSelectData
when '= Add_List_Entry'
* Add an entry to the list view
fetch FIELDS(#XG_Ident) FROM_FILE(PSLMST) WITH_KEY(#EMPNO)
if status *OKAY
* Start an instance list update
Invoke Method(#avListManager.BeginListUpdate)
* Set up the visual Identifier(s)
Change #UF_VisID1 #EMPNO
Change #UF_VisID2 #GIVENAME
Use BConcat (#UF_VisID2 #SURNAME) (#UF_VisID2)
* Add instance details to the instance list
```

```
Invoke #avListManager.AddtoList Visualid1(#UF_VisID1)
Visualid2(#UF_VisID2) AKey1(#DEPTMENT) AKey2(#SECTION)
AKey3(#EMPNO) ACOLUMN1(#PHONEHME) ACOLUMN2(#ADDRESS1)
NCOLUMN1(#POSTCODE)
* Instance list updating has been completed
Invoke Method(#avListManager.EndListUpdate)
endif
```

when '= Update_List_Entry' * Update an entry that already exists in the instance list fetch FIELDS(#XG_Ident) FROM_FILE(PSLMST) WITH_KEY(#EMPNO) if status *OKAY * Start an instance list update Invoke Method(#avListManager.BeginListUpdate) * Set up the visual Identifier(s) Change #UF_VisID1 #EMPNO Change #UF_VisID2 #GIVENAME Use BConcat (#UF_VisID2 #SURNAME) (#UF VisID2) * Add instance details to the instance list Invoke #avListManager.UpdateListEntryData Visualid1(#UF_VisID1) Visualid2(#UF_VisID2) AKey1(#DEPTMENT) AKey2(#SECTION) AKey3(#EMPNO) ACOLUMN1(#PHONEHME) ACOLUMN2(#ADDRESS1) NCOLUMN1(#POSTCODE) * Instance list updating has been completed Invoke Method(#avListManager.EndListUpdate) endif

when '= Delete_List_Entry'
Invoke Method(#avListManager.BeginListUpdate)
* Remove instance details from the instance list
Invoke #avListManager.RemoveFromList AKey1(#DEPTMENT)
AKey2(#SECTION) AKey3(#EMPNO)
Invoke Method(#avListManager.EndListUpdate)
endcase

Endroutine End_Com

Subfiles/Browselists

From time to time you will need to create scripts that access 5250 subfiles. There are three common approaches used to do this:

Movie Not Using a Datagrid Control - 1 minute

Movie Using Subfile Accessor - 5 minutes

Movie Subfile Direct Access - 2 minutes (If you need to use subfile direct access then please review this movie first.)

After this following additional script samples may be useful to you in different situations:

Script for Locating an Entry in a Subfile/Browselist

Script for Locating and Selecting an Entry in a Browselist or Subfile by Positioning the Cursor

Script for Locating an Entry when no Positioning is Available

Script for Locating an Entry when no Positioning is Available and the List has more than One Page

Also See SUBFILE_ACCESSOR Object

Not Using a Datagrid Control - 1 minute

•

Play Movie to learn how to manage subfiles/browselists without datagrid control or read the Movie Summary.

Movie Summary

For movie Not Using a Datagrid Control - 1 minute.

When a 5250 subfile is presented as a data grid on a junction screen the first question you should always ask: "*Can my script always ensure that the data I am interested in occurs in the first entry in the subfile*?"

If the answer is yes, then the easiest way to access the content of the 5250 subfile data grid is to disable the grid and treat the entry (browselist cell) as a field:

Disable the grid

In newlook Designer, open the properties of the Form object by double-clicking on the form. Locate the Recognition UseGrids property and set it to False.

Close the designer and save your changes.

Now newlook recognizes the browselist cells as normal fields.

Name the first cell as a field

Open the Designer again. The subfile area is now presented as a series of simple text boxes and labels, rather than as a data grid.

Right-click the first text box to bring up its properties and give it a name using the Name property, for example *uSelectEmployee*.

The script can now put a value to the field just as if it was a normal field:

SETVALUE("uSelectEmployee", "8"); SENDKEY(KeyEnter);

Using Subfile Accessor - 5 minutes

•

Play Movie to learn how to use the subfile accessor or read the Movie Summary.

Movie Summary

For movie Using Subfile Accessor - 5 minutes.

A typical Work With screen has a subfile/browselist of objects (in this example Employees), each with an option field.

In RAMP a subfile is displayed as a data grid.

In this tutorial we want to enter option "8" in front of a specific entry in the data grid:

Name the data grid

Open the screen in newlook Designer, click on the grid and specify EMPLOYEE_LIST as the Name property in the Misc section.

Note that the data grid and the subfile have different numbers of columns and rows

- A data grid starts column and row numbering from 0 and includes the header rows
- Column numbers are not necessarily sequential in a data grid
- Columns have names based on the text in the column headings

To select an employee in the grid using SUBFILE_ACCESSOR

Create the SUBFILE_ACCESSOR object: var SFL = new SUBFILE_ACCESSOR ("EMPLOYEE_LIST");

Put an "8" somewhere in column "Opt": SFL.SetSelectionColumnName("Opt","8");

Then locate the employee that has "A0090" in column "EmployNumber": SFL.SetSearchColumnName("EmployNumber","A0090");

Put the value into the subfile: SFL.SelectSubfileEntry();

Lastly, destroy the subfile object: SFL.Dispose();

Do not write subfile code manually

Instead see how to create a subfile script Using the Scripting Pop-up Menu and then modify it as required.

Find out row and column names

Use the Probe button which appears on RAMP screens.

Also see SUBFILE_ACCESSOR Object.

Subfile Direct Access - 2 minutes

•

Play Movie to learn how to directly access subfiles or read the Movie Summary.

Movie Summary

For movie Subfile Direct Access - 2 minutes.

This tutorial describes the subfile direct access foundation script.

Create the script Using the Scripting Pop-up Menu. The script reads all subfile pages, and for each page it reads all columns and for each column it reads all the cells.

By modifying this script you should be able to perform any required subfile operation.

Foundation Script

```
{
var strDataGridName = "xxxxxxxxx"; /* Specify the data grid name here */
var flagAnotherPageExists = true; /* Another subfile page exists */
/* Loop through all subfile pages */
do
{
var intColumnCount = TONUMBER(GETVALUE(strDataGridName +
".Columns.Count"));
var intRowCount = TONUMBER(GETVALUE(strDataGridName +
".RowCount"));
var intColumn, intRow = 0;
/* Iterate over the current subfile page */
for (intColumn = 0; intColumn < intColumnCount; intColumn++) /* Iterate
through the columns */
{
var strColumn = intColumn.toString();
var strColumnName = GETVALUE(strDataGridName + ".Columns(" +
strColumn + ").Name");
TRACE("Column number " + strColumn + " is named \"" + strColumnName +
"\"");
/* Iterate through the cells in the column */
for (intRow = 0; intRow < intRowCount; intRow++) /* Iterate through the rows
for a column */
{
var strRow = intRow.toString();
var strRowCellValue = GETVALUE(strDataGridName + ".Columns(" +
strColumn + ").Cells(" + strRow + ").Text");
TRACE(" in row " + strRow + " it contains the value \"" + strRowCellValue +
```

"\"");

} /* end iterating the rows for a column */

} /* end iterating the columns */

/* Proceed (or not) to the next subfile page based on the marker */

flagAnotherPageExists = (GETVALUE(strDataGridName + ".Marker") != ""); if (flagAnotherPageExists) { TRACE("Scrolling to next page.");

SENDKEY(KeyPageDown); }

else { TRACE("End of subfile encountered"); }

} while(flagAnotherPageExists); /* Loop around and process the next subfile
page */

} /* NOTE: This script is dependent on the use of newlook for 5250 access */

To test you have the row you want

To work with subfile entries you need to compare your search value with the text in the cell. To read the text in a cell:

strCell = StrGridName+".Columns("+strColumn+").Cells("+strRow+").Text;

And put the value into a GETVALUE function: StrRowCellValue = GETVALUE(strCell);

To set the value of a cell: SETVALUE(strCell, "some valid value");

Locating and Selecting an Entry in a System i Subfile/LANSA Browselist

Many 5250 applications use System i Subfiles or LANSA Browselists to allow the end-user to select the object they want to work with.

In a modernized RAMP application that uses filters and the instance list the subfiles/browselists typically become superfluous so you will probably want to hide them. You do this by using a script that automatically locates the entry in the subfile/browselist without the user having to interact with the screen.

Script for Locating an Entry in a Subfile/Browselist

Script for Locating and Selecting an Entry in a Browselist or Subfile by Positioning the Cursor

Script for Locating an Entry when no Positioning is Available

Script for Locating an Entry when no Positioning is Available and the List has more than One Page

Script for Locating an Entry in a Subfile/Browselist

This script locates an entry in a browse list.

The position of the entry in the list is determined by the value entered in an input field. Then, perform an action on the selected the entry.

In this script:

 \cdot utxtEmpno is the name given to the field to position to in the form using newlook Designer

 \cdot uDataGrid Is the name given to the browselist/subfile in the form using newlook Designer

• The first column in the data grid contains the Option field.

- The second column in the data grid contains the value to position to.
- The VisualID1 of the Instance List has the employee number
- \cdot Use option 2 to change the details of an employee selected in the Instance List.

```
/* Set the value of a variable to that of the selected Employee */
var strCompare = objListManager.VisualId1[0];
```

```
/* Set the value of the field in the form to that of the selected Employee */
```

SETVALUE("utxtEmpno", strCompare);

```
/* Send an Enter key to cause the list positioning */
```

SENDKEY(KeyEnter);

```
/* Get the number of rows in the data grid */
```

```
var intRowCount = GETVALUE("uDataGrid.RowCount");
```

/* Traverse the rows and compare the value of the second column with the one to position to. */

```
for (var intRowNo = 0; intRowNo < intRowCount; intRowNo++)
{</pre>
```

```
if ( GETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(1).Text") ==
strCompare)
```

```
{
    {
        /* Type a 2 next to the first entry and press enter */
        SETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(0).Text", 2);
        SENDKEY(KeyEnter);
        break;
    }
}
```

Script for Locating and Selecting an Entry in a Browselist or Subfile by Positioning the Cursor

Sometimes entries in browselists/subfiles are selected by positioning the cursor on the desired row and pressing Enter.

To position the cursor in a desired row/column you can use the provided SETCURSOR() interface. Please refer to <setcursor> to find out details of the uses of SETCURSOR().

SETCURSOR receives a row, column and row offset position. The latter is the row number of the first entry in the browselist/subfile. Note that newlook's Data Grids counts the column headings as rows. To find out the row position of a browselist or subfile, run the application in green screen mode until you reach the desired list. You should be able to see the row/column positions on the bottom right of the screen. Move the cursor until the column heading's top most piece of text. That will be the row offset value to pass.

```
var strCompare = objListManager.VisualId1[0];
/* This is the row number where the first row entry in the list. Column headings
are counted as rows */
var intListOffset = 3;
var flagPageDown = true;
while (flagPageDown)
ł
intRowCount = GETVALUE("uuDataGrid.RowCount");
for (var intRowNo = 0; intRowNo < intRowCount; intRowNo++)
ł
/* The value we want to compare with is in the first cell */
   if (GETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(0).Text") ==
strCompare)
   {
     SETCURSOR(intRowNo, 10, intListOffset);
     SENDKEY(KeyEnter);
     flagPageDown = false;
     break;
   }
 if (flagPageDown)
   if (GETVALUE("uDataGrid.Marker") != "") SENDKEY(KeyPageDown);
```

```
else { flagPageDown = false; alert("page down is false"); }
}
```

Script for Locating an Entry when no Positioning is Available

A variation on the previous script is to find an entry in a list where there is no positioning available and we do not know which column holds the sought after value. In such scenario, for each row we'd have to traverse each cell. Note that for big lists this can be a time consuming task:

```
for (var intRowNo = 0; intRowNo < intRowCount; intRowNo++)
{
    /* Get the number of cells in this row. */
    var intColCount = GETVALUE("uDataGrid.ColCount");
    for (var intColNo = 0; intColNo < intColCount; intColNo++)
    {
        if ( GETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(" + intColNo +
").Text") == strCompare)
        {
            /* Type a 2 next to the first entry and press enter */
            SETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(0).Text", 2);
            SENDKEY(KeyEnter);
            break;
        }
    }
}</pre>
```

Note that the above script will only handle the first page in the list.

Script for Locating an Entry when no Positioning is Available and the List has more than One Page

The previous example showed how to handle a list with only one page. If a list has more than one page, we have to page down until the end of the list.

Therefore, the for loop should be enclosed within another loop to be executed while we haven't reached the end of the list. Note that in this case, the row count must be retrieved for each page.

```
var flagPageDown = true;
while (flagPageDown)
ł
intRowCount = GETVALUE("uDataGrid.RowCount");
for (var intRowNo = 0; intRowNo < intRowCount; intRowNo++)
{
 /* Get the number of cells in this row. */
 intColCount = GETVALUE("uDataGrid.ColCount");
 for (var intColNo = 0; intColNo < intColCount; intColNo++)</pre>
 {
   if ( GETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(" + intColNo +
").Text") == strCompare)
   {
     /* Type a 2 next to the first entry and press enter */
     SETVALUE("uDataGrid.Rows(" + intRowNo + ").Cells(0).Text", 2);
     SENDKEY(KeyEnter);
 /* Reset the flag to cause the while loop to end */
     flagPageDown = false;
     break:
   }
 }
 if (flagPageDown)
 {
    /* Get the value of the newlook indicator that tells us whether there is
another page in the list. If not, reset the flag to cause the while loop to end */
   if (GETVALUE("uDataGrid.Marker") != "") SENDKEY(KeyPageDown);
   else flagPageDown = false;
 }
}
```

Handling Pop-Ups

When newlook recognizes a border pattern on a 5250 screen, it creates a pop-up window. You are not compelled to present these pseudo pop-ups as a real windows, and we strongly recommend you do not define them as destination screens.

During Navigation Pop-up as Destination When Triggered by Button Click or Function Key Press Forcing a Pop-Up to Front How to Turn Pop-Ups into Full Screens

Also see FORCE_POPUP_REFRESH Function.

During Navigation

Situation

RAMP navigates through a pop-up on its way to a destination screen.

Expected Behavior

Windows: the pop-up is not visible during navigation. Browser: the pop-up is visible during navigation.

Remarks

You cannot interact with newlook when it is hidden in the browser.

Pop-up as Destination

We recommend you absolutely avoid using 5250 pop-up screens as RAMP destination screens.

Having a pop-up jump out of command tab would be very unusual. We recommend you instruct newlook to present the 5250 pop-up as a full screen. See How to Turn Pop-Ups into Full Screens.

When Triggered by Button Click or Function Key Press

Situation

Ramp navigates to a destination. The user clicks on a VLF push button or presses a function key to display a pop-up.

Expected Behavior

Windows and Browser: the pop-up will not be visible and the main screen becomes grayed out and input incapable.

Remarks

In this case the pop-up is hidden behind the main screen. See Forcing a Pop-Up to Front.

Forcing a Pop-Up to Front

If you need to force a pop-up window to front (for example if you absolutely have to present a 5250 pop-up window as a destination screen), using FORCE_POPUP_REFRESH Function will not help you because it only turns the automatic force-to-front logic off or on. It does not change when the logic is used.

In these special cases you will have to invoke the force logic yourself, probably at the end of your destination pop-up screens INVOKE script.

The easiest way to do this is by executing the JavaScript function VF_SY120_FORCE_POPUP_REFRESH(). This will cause the current 5250 pop-up to come to the front.

If you want to force a pop-up which is triggered by a button click or function key press to front, do it like this:

SENDKEY(KeyF10);

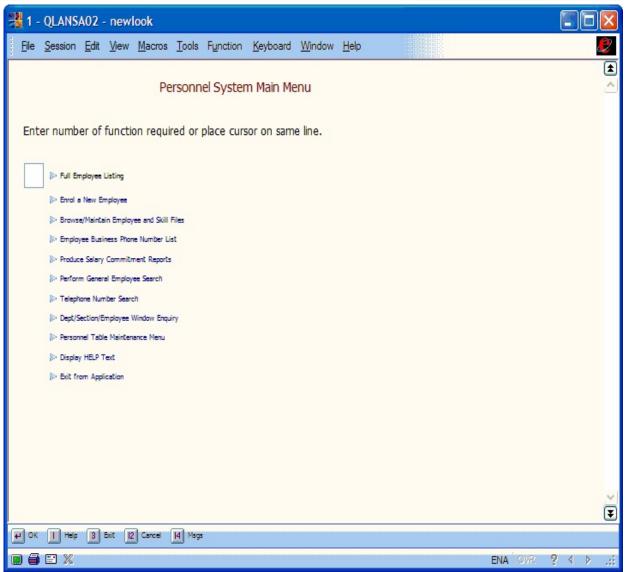
VF_SY120_FORCE_POPUP_REFRESH();

How to Turn Pop-Ups into Full Screens

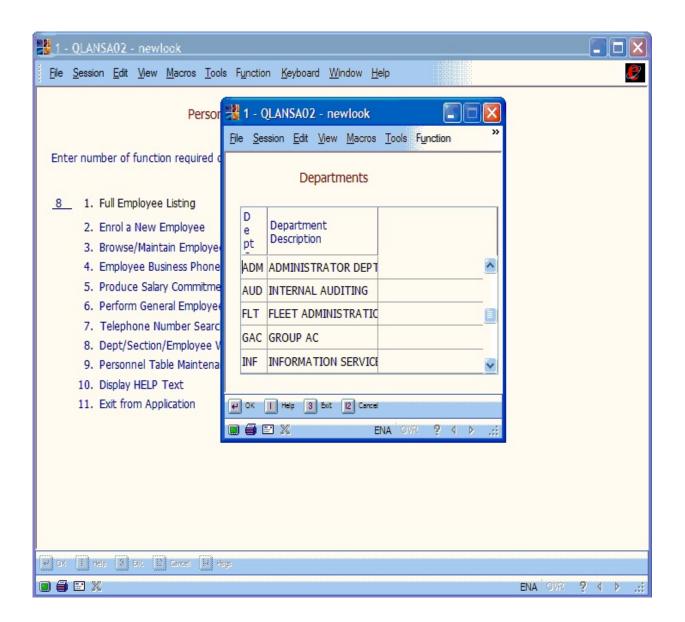
This example turns a pop-up into a full screen. It uses the Department pop-up called using option 8 (Dept/Section/Employee Window Enquiry) from the Personnel System's main menu.

1. Start newlook, connect to your System i and sign on. On the command line, invoke the Personnel System's main menu. For example,

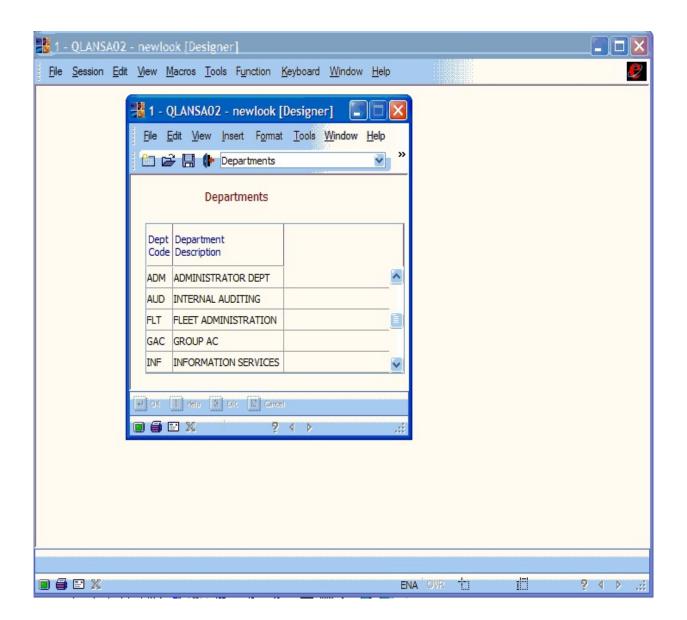
lansa run pslsys partition(dem)



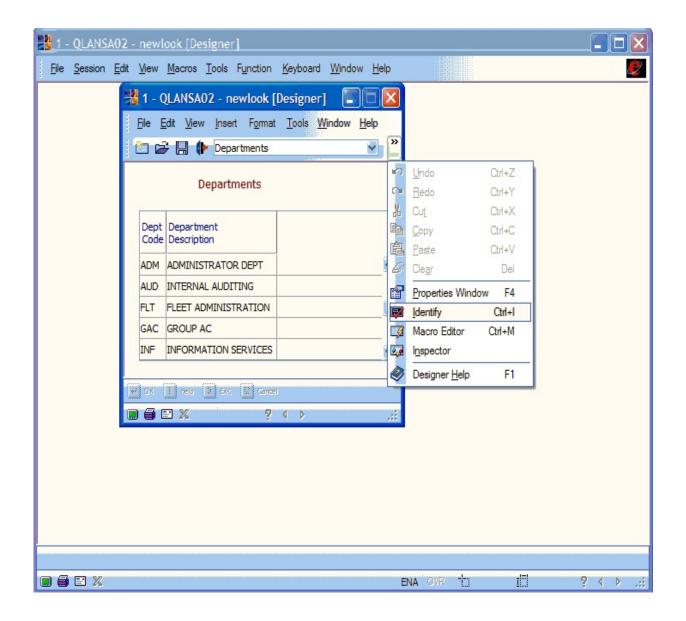
2. Use option 8 to display the Departments prompter:

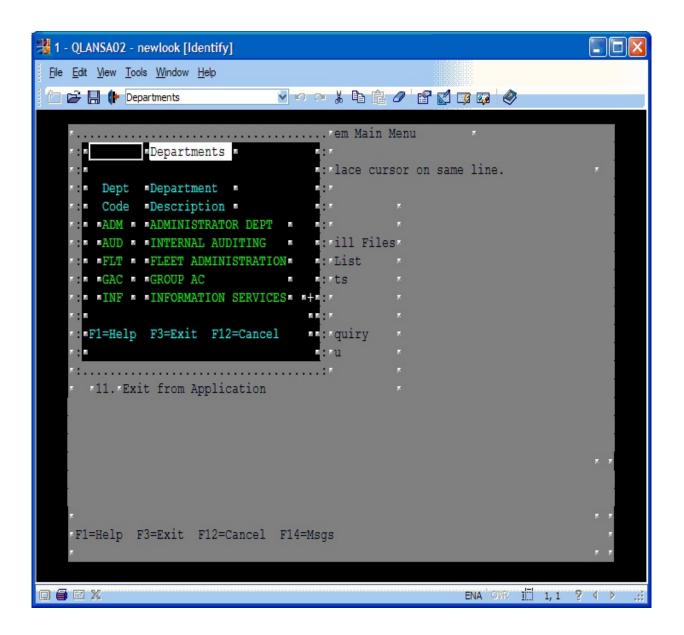


3. Press Ctrl+D to start newlook Designer. The screen should look something like this:

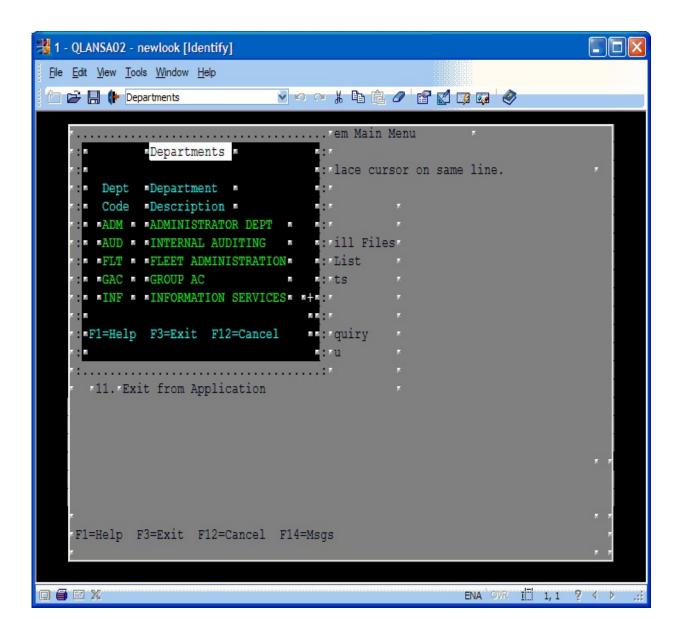


4. Click on Identify to work in newlook Identify mode:





5. Use the mouse to select the entire grey area:



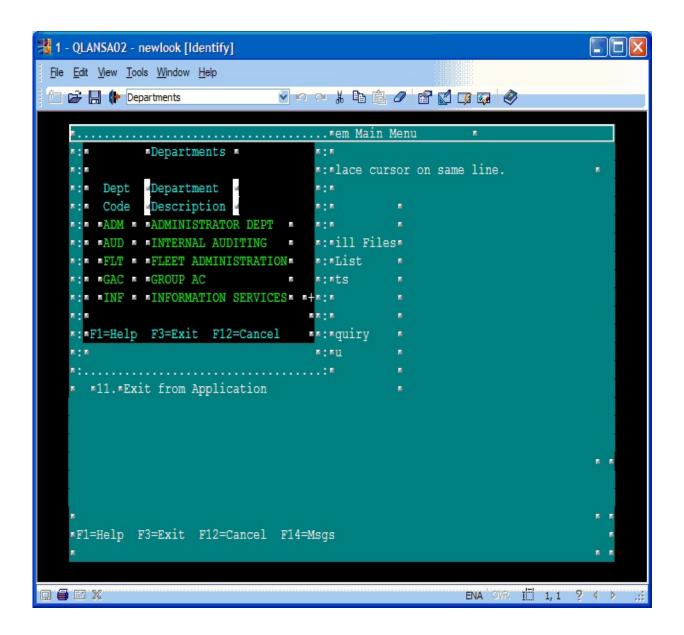
6. Right click anywhere on the selected area and select the Set Window Area option in the Screen submenu:

Normal State	
<u>File Edit Vi</u> ew <u>T</u> ools <u>Wi</u> ndow <u>H</u> elp	
🖆 🚅 🔚 (🕪 Departments 💿 🕫 🕫 🖉 😭 🐼 🐼 🐼	
<pre>*:* * Departments * * * * * * * * * * * * * * * * * * *</pre>	
<pre>>: Dept Department ></pre>	
Code Description	
★: ■ ADM ■ ADMINISTRATOR DEPT ■ ■: * ✓ Screen ID	
*: • • AUD • • INTERNAL AUDITING • • •: •ill Fi. Unknown	
*: • • FLT • • FLEET ADMINISTRATION • •: • List Ignore	
K ■ GAC ■ ■GROUP AC ■ ■ K: Kts Screen Function Ke	ey .
INF ■ INFORMATION SERVICES ■ ■+ ■: ■ Field Message	
: Menu Message M *:*F1=Help F3=Exit F12=Cancel Message M	
Miscellaneous 🕨	
<pre>// / / / / / / / / / / / / / / / / / /</pre>	v Area
	la de la companya de
	म म
	7 7
F1=Help F3=Exit F12=Cancel F14=Msgs	7
	N N
ENA OVR 1	□ 1, 1 ? < ▷ .;;

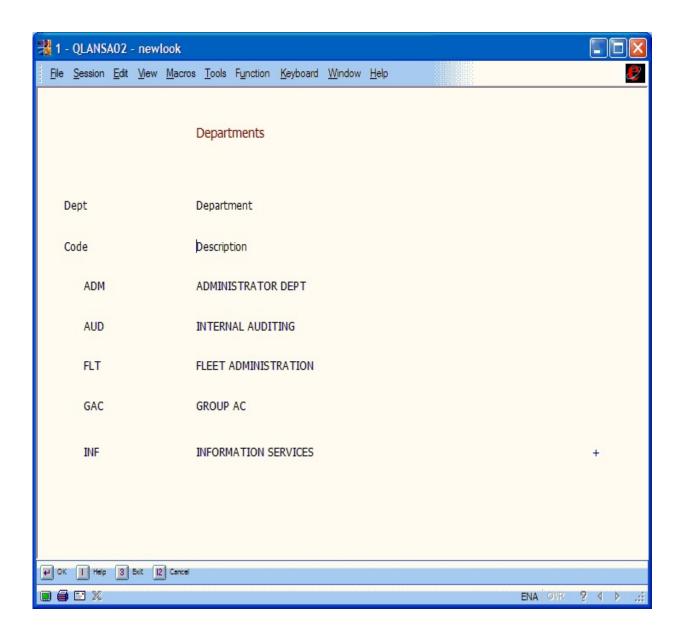
Hansa - QLANSA02 - newlook [Identify]	
<u>Fi</u> le <u>E</u> dit <u>V</u> iew <u>T</u> ools <u>W</u> indow <u>H</u> elp	
🖆 🚅 🔚 🌗 Departments 🔍 🔊	·· # Pa Ca 2 ' Ca Ca Ca Ca 🛷
	¤em Main Menu 🛛
ala aDepartments a	n ; n
# 1 #	<pre>n:slace cursor on same line.</pre>
•:• Dept •Department •	*:*
•: Code •Description •	*;* *
*: * ADM * ADMINISTRATOR DEPT *	
<pre>%:* #AUD # #INTERNAL AUDITING # #:* #FLT # #FLEET ADMINISTRATION*</pre>	
	n:nts
A: A BINE A BINFORMATION SERVICESA A	
*: F1=Help F3=Exit F12=Cancel	
a ; a	N:NU N
a:	
 \$11. Exit from Application 	×
	n n
F1=Help F3=Exit F12=Cancel F14=	Msgs
	80 B
	ENA O'R İ 1, 1 💡 4 🕨 🛒

7. Now select everything surrounding the pop-up area, right click on the selected area and select Ignore. Note that you may have to do more than one selection to cover the whole of the surroundings. Make sure to have something selected as the pop-up screen id. In the example below, the id is the attribute bytes enclosing the Department and Description subfile headers:

Normal State (1997) 1 - QLANSA02 - newlook [Identify]		
<u>File Edit View T</u> ools <u>W</u> indow <u>H</u> elp		
🛍 🚅 🔚 🌗 Departments 🔍 🔹	v • # 🖻 🖻 🖉 😭 🐼 🐺 🤣	
	*em Main Menu *	
<pre>N:N NDepartments N</pre>		
≋:¤ ¤:¤ Dept ⊴Department ⊴	<pre>*:*lace cursor on same line. *:*</pre>	M
¤:■ Dept ⊴Department ⊴ ¤:■ Code ⊴Description ⊴	n n n n	
: *ADM * *ADMINISTRATOR DEPT *		
A: AUD A AINTERNAL AUDITING		
*: * FLT * *FLEET ADMINISTRATION*	∎:∎List ■	
	Screen ID	
*: * *INF * *INFORMATION SERVICES*	• Shalonin	
	Ignore	
<pre>N: F1=Help F3=Exit F12=Cancel N: N</pre>	Screen	·
*:	Field	
#11.#Exit from Application	Subfile Subfile	
	Command	• I
	Miscellaneous	; • I
	Group Fields	
		82 B
		n n
F1=Help F3=Exit F12=Cancel F14	4=Msgs	
		N N
	ENA 'OR 🗐 1,40 ?	∢ ⊳ .∷



8. Close Identify. The screen should now look something like this:



The pop-up has now been converted into a full screen.

Script Functions

This section describes the shipped RAMP JavaScript functions you can use in your scripts.

Note that these functions are case sensitive, so be careful to use exactly the same case as shown when writing scripts.

Get and Set Values of fields on RAMP Screens	
Set a field on a RAMP screen to a value	SETVALUE Function
Get the value from a field on a RAMP screen	GETVALUE Function
Screen Functions	
Emulate pressing a key.	SENDKEY Function
Check that RAMP is showing a screen	CHECK_CURRENT_FORM Function
Signal to the Framework to close the current screen	AVCLOSEFORM Function
Hide the current screen with an optional message	HIDE_CURRENT_FORM Function
Get the Form Name of the current RAMP screen	CURRENT_FORM Function
Position the cursor in a given row and column of the screen	SETCURSOR Function
Searches the label fields on the form for a	GET_MENU_OPTION_NUMBER Function

menu option by name	
Function keys to send when an unknown form appears during RAMP navigation	ADD_UNKNOWN_FORM_GUESS Function
Overcome problem of a pop-up which is invoked from a button click or function key press in a destination screen being hidden behind the main screen	FORCE_POPUP_REFRESH Function
Set the current command handler caption to a new value	SET_HANDLER_CAPTION Function
Dynamically enable or disable a Destination's button or 5250 function key	SETKEYENABLED Function
Message Functions	
Issue a message as an alert	ALERT_MESSAGE Function
Clear all messages currently in the stack	CLEAR_MESSAGES Function
Issue a fatal message	FATAL_MESSAGE Function
Issue a message	MESSAGE Function
Stop the Framework from shutting down when a fatal navigation error occurs	FATAL_MESSAGE_TYPE Function

Other Functions	
Signal an event to filters and RAMP screens	AVSIGNALEVENT Function
Add run time information to the trace panel	TRACE Function
Run a newlook Macro	RUNMACRO Function
Navigate to a Junction screen	NAVIGATE_TO_JUNCTION Function
Show a user defined prompter form for a field	HANDLE_PROMPT Function
Navigate to a Destination screen	NAVIGATE_TO_DESTINATION Function
Navigate to a previously shown Destination screen	NAVIGATE_TO_PREV_DESTINATION Function
Returns the leading numbers from a string	STRIP_LEADING_NUMBERS Function
Defines a string by a unique number for use by other scripts	ADD_STRING Function
Sets the message to show when the Framework locks up	SET_LOCK_MESSAGE Function
Returns the string for a given string identification number	STRING Function
Call this function to	SET_UNKNOWN_LOCKING Function

override the Lock Framework when unknown 5250 form is displayed session property	
Set the focus to a field on the current screen	SETFOCUS Function
Get the name of the field with focus on the current screen	GETFOCUS Function
Return a DataGrid as a string	MAKESUBFILEINTOSTRING Function
newlook Function Key Bar Functions	
Causes all screens not defined to RAMP to show the newlook toolbar buttons	OVERRIDE_BUTTONS_UNDEFINED_SCREENS Function
Multilingual Caption Functions	
Assigns a new caption for a function key on a particular screen	OVERRIDE_KEY_CAPTION_SCREEN Function
Assigns a new caption for a function key on any screen	OVERRIDE_KEY_CAPTION_ALL Function
Virtual Clipboard Access Functions	
Save an alphanumeric or numeric value in the Framework virtual clipboard.	AVSAVEVALUE Function
Restores an	AVRESTOREAVALUE and AVRESTORENVALUE

alphanumeric or numeric value from the Framework virtual clipboard	Function
Real Clipboard Access Functions	

Copy a string to the user's clipboard

COPYTOCLIPBOARD Function

SETFOCUS Function

Set the focus to a field on the current screen.

Syntax

SETFOCUS(sControlName);

Parameters

sControlName	Required. String that contains the name of the field
	that focus is to be set to.

Return Value

None

Example

SETFOCUS("utxtSurname");

Notes

The field (or other control) name is the name assigned to the field in Newlook Identify.

This function uses a new macro in VF_MACRO.sid, called

VF_Macro.VF_Set_Active_Control. So before this function can be used, the latest shipped VF_MACRO.sid must be merged into your Newlook.sid file.

GETFOCUS Function

Get the name of the field with focus on the current screen.

Syntax

GETFOCUS();

Parameters

None

Return Value

	String that contains the name of the field that has focus. It is "" if no control is found, or if the
	Control with focus does not have a name.

Example

var strValue = GETFOCUS();

Notes

The field (or other control) name is the name assigned to the field in Newlook Identify.

MAKESUBFILEINTOSTRING Function

Return a DataGrid as a string.

Syntax

MAKESUBFILEINTOSTRING(sDataGridName, All_Pages, Include_Heading, sHeader_Column_ Numbers, sDetail_Column_Numbers)

Parameters	,
------------	---

sDataGridName	Required. String that contains the name of the data grid that is to be returned as a string.
All_Pages	Optional. Boolean that tells the function to page down through all pages in the subfile. Defaults to false.
Include_Heading	Optional. Boolean that tells the function to include the subfile header rows in the returned string. Defaults to true
sHeader_Column_Numbers	Optional. String that contains the column numbers of the header rows to show, delimited by commas. Default is all non-blank columns
sDetail_Column_Numbers	Optional. String that contains the column numbers of the detail rows to show, delimited by commas. Default is the columns used for the header columns

Return Value

sSubfileAsString	Required. String that contains the subfile as a tab	
	delimited string.	

Examples

/* Get just the currently visible page of the subfile named uDataGrid */

/* Get the header rows, */

/* and paste them all onto the clipboard. */

/* Only show the header cells in columns 0,2,3,4 and 5 */

/* Only show the detail cells in columns 1,2,3,4 and 6 */

COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", false, true, "0,2,3,4,5", "1,2,3,4,6"));

/* Get all the pages of the subfile named uDataGrid */

/* Get the header rows, */

/* and paste them all onto the clipboard. */

/* Only show the header cells in columns 0,2,3,4 and 5 */

/* Only show the detail cells in columns 1,2,3,4 and 6 */

COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", true, true, "0,2,3,4,5", "1,2,3,4,6"));

/* Get all the pages of the subfile named uDataGrid */

/* Get the header rows, */

/* and paste them all onto the clipboard. */

/* Only show the header cells in columns 0,2,3,4 and 5 */

/* Only show the detail cells in columns 0,2,3,4 and 5 */

COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", true, true, "0,2,3,4,5"));

/* Get all the pages of the subfile named uDataGrid */

/* Get the header rows, */

/* and paste them all onto the clipboard. */

/* Show the non-blank header cells */

/* Only show the detail cells in columns 1,2,3,4 and 6 */

COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", true, true, null,"1,2,3,4,6"));

/* Get all the pages of the subfile named uDataGrid */

/* Get the header rows, */

/* and paste them all onto the clipboard. */

/* Show the non-blank header cells */

/* Only show the detail cells in columns with non-blank headers */

COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", true, true));

/* Get all the pages of the subfile named uDataGrid */

/* Get all the detail rows, but don't get the header rows, */

/* paste them onto the clipboard. */

/* Only show the detail cells in columns 1,2,3,4 and 6 */

COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", true, false, null, "1,2,3,4,6"));

Notes

Use the probe screen report to ensure that Newlook is displaying the subfile as a datagrid, and to determine the column numbers of the header and detail cells that you want to show.

Can be used in combination with function COPYTOCLIPBOARD to allow the user to paste a subfile into their own excel spreadsheet.

COPYTOCLIPBOARD Function

Copy a string to the user's clipboard.

Syntax

COPYTOCLIPBOARD(sString);

Parameters

sString	Required. String that contains the data to be
	copied to the user's clipboard.

Return Value

None

Examples

```
COPYTOCLIPBOARD("ABC");
```

```
COPYTOCLIPBOARD(MAKESUBFILEINTOSTRING("uDataGrid", true, true, "0,2,3,4,5", "1,2,3,4,6"));
```

/* Copy to a spreadsheet */

```
var MyString = "";
var TAB_Char = "\x09";
var End_Of_Line_Char = "\x0D\x0A";
```

```
MyString = "Line 1 Cell 1" + TAB_Char + "Line 1 Cell 2" +
End_Of_Line_Char;
MyString += "Line 2 Cell 1" + TAB_Char + "Line 2 Cell 2" +
End_Of_Line_Char;
COPYTOCLIPBOARD(MyString);
```

Notes

This function can be used to allow the user to copy data to their real clipboard, for pasting into Word documents or spreadsheets

FATAL_MESSAGE_TYPE Function

Use this function when you don't want the Framework to shut down when a fatal navigation error occurs.

Syntax

FATAL_MESSAGE_TYPE(sType)

Parameters

sType	Optional. String that contains the message type:
	FATAL (default) – in end user mode, the framework will shut down.
	HIDE – the RAMP command tab will hide Newlook 5250 and show the error.
	INFO – the error message will be routed to the Framework message area.

Return Value

None

Example

FATAL_MESSAGE_TYPE("HIDE");

SET_UNKNOWN_LOCKING Function

We strongly recommend you do not to use this function because the default locking behaviour is correct in most situations. If you think you need to use this function please contact your support representative.

Call this function to override the *Lock Framework when unknown 5250 form is displayed* session property. See *Session Details* for more information.

Syntax

SET_UNKNOWN_LOCKING(boolean)

Parameters

Boolean. One of the following possible values:

true	Apply a lock to the framework when an unknown 5250 form is encountered.
false	Don't lock the framework when an unkown 5250 form is encountered.

Return Value

None.

Remarks

Invoke only once per session from the sign on script. Dynamically changing this value may cause undesirable results.

SETKEYENABLED Function

Dynamically enable or disable a destination's button or 5250 function key. This function overrides the destination's function key enablement, for the duration of the logged on 5250 session. The override will impact all future displays of the destination screen.

Syntax

SETKEYENABLED (sDestinationName,sKeyName,bEnableVLF,bEnableNL)

Parameters

sDestinationName	Required. A string that contains the name of a Destination.
sKeyName	Required. String that contains the name of the key. See Function Key Names for SENDKEY Function.
bEnableVLF	Optional. Boolean. Set to true to show the button, false to hide it, null to ignore.
bEnableNL	Optional. Boolean. Set to true to enable the 5250 function key, false to disable it, null to ignore.

Return Value

None

Example

The Destination named uDisplayEmployee was set up to Show the prompt button but disable the F4 5250 function key.

To override those settings to the reverse:

SETKEYENABLED("uDisplayEmployee", KeyF4, false,true);

To leave the original setting for the button but enable the F4 function key as well:

SETKEYENABLED("uDisplayEmployee", KeyF4, null,true);

SETVALUE Function

Set the content of a field on a 5250 screen to a value. The field may be identified by name or by its order on the screen.

Syntax

Setting by Name - SETVALUE(sVariable , sValue) Setting by Order - SETVALUE(___Field , sOrder, sValue)

Parameters

Setting by Name:

sVariable	Required. String that contains the RAMP field name.
sValue	Required. String that contains the value to set the field to.

Setting by Order:

Field	Special valueField (with two underscores) indicates that a field (ie: a simple text area) on the form is to be set.
sOrder	The order of the field on the form starting from 1. Special valuesLast andFirst (again with two underscores) may be used. Note the order is that of the fields on the form, not of all the controls on the form (eg: labels, combo boxes, etc).
sValue	Required. String that contains the value to set the field to.

Return Value

None

Remarks

To set a value of a field on a screen by name, the field must be given a name in the newlook Designer.

The use of field identification by order is more likely to be impacted by form layout changes than by using a name.

The initial setting of a field by order is more expensive to execute than by name, however screen field order details are cached so that the subsequent access is faster. The caching logic assumes that the relative order of a field on any particular screen will not change within a signed on 5250 session.

Examples

SETVALUE("utxtSignOn", objUser.Name);

SETVALUE("utxtPassword",objUser.Password);

SETVALUE("utxtSelectionOrCommand","90");

SETVALUE("utxtTransaction","MOV");

SETVALUE(___Field,___First,"xxx"); /* Set first field */

SETVALUE(___Field,___Last,"7.45"); /* Set last field */

SETVALUE(___Field,1,"Hello World"); /* Set field number 1 */

SETVALUE(___Field,6,"ADM"); /* Set field number 6 */

GETVALUE Function

Get the value from a field on a RAMP screen.

Syntax

GETVALUE(sVariable)

Parameters

sVariable Required.String that contains the newlook field name.

Return Value

String. Returns the value of the field, as a string:

Example

MyString = GETVALUE("utxtSignOn");

SENDKEY Function

Emulates the pressing of a key.

Syntax

SENDKEY(sKeyName)

Parameters

SKeyName	Required.String that contains the name of the key. See
	Function Key Names for SENDKEY Function.

Return Value

None

Example

SENDKEY(KeyEnter);

CHECK_CURRENT_FORM Function

Check that RAMP is showing a screen.

Syntax

CHECK_CURRENT_FORM(sFormName [, sMessageText1] [, sMessageText2] ...)

Parameters

sFormName	Required. String that specifies the Name of the Form
sMessageText1	Optional. String that contains the first message to be issued.
sMessageText2	Optional. Other strings that are to be concatenated with the first message string (a separator space is automatically added between each string).

Return Value

Boolean. Returns one of the following possible values:

true	The form currently shown has the form name specified.
false	The form currently shown does not have the form name specified.

Remarks

Used for checking whether the script or user has progressed to a particular screen, or has stopped at an earlier screen.

The Form name for a RAMP screen is found by working with the screen in newlook in Designer mode and setting the Name property of the Form object.

If the CHECK_CURRENT_FORM returns false, the function will also automatically hide the Current RAMP screen and display the message provided.

If the script wants to test that the expected screen has arrived, and yet still display the current screen if it hasn't, it should not use function CHECK_CURRENT_FORM, but instead use

```
if (CURRENT_FORM() == "My_Form");
```

When you are writing scripts that handle validation errors on a screen, you usually want the current screen to be displayed even if a validation error occurred and the user has not progressed to the expected next screen. So in this situation you should not use CHECK_CURRENT_FORM.

Example

if (!(CHECK_CURRENT_FORM("uItemMasterBrowse","Unable to navigate
to form uItemMasterBrowse"))) return;

AVCLOSEFORM Function

Signals to the Framework to close the current form.

Syntax

AVCLOSEFORM()

Parameters

None

Return Value

None

Remarks

If a RAMP screen is running as a separate form, and needs to be closed automatically after completing, use AVCLOSEFORM.

Ensure that the current form is a form known to the Visual LANSA Framework, at the point the AVCLOSEFORM is issued, and that there is a valid Return Script for this Junction or Destination. This will allow the Framework to navigate back to sign off and end the session cleanly.

Example

/* Close this command handler, since the Delete is now done */

/* We should ensure we are on a Junction or Destination at this point */

/* so that the Framework can cleanly navigate the newlook session to sign off */ AVCLOSEFORM();

HIDE_CURRENT_FORM Function

Hides the current form and displays an optional message.

This function is used to hide the current 5250 screen from the users and to prevent them from manually interacting with it.

For example, a script that performed a 5250 sub-file search and failed to find an expected product number might do this:

HIDE_CURRENT_FORM("Product number", strProductNumber, "could not be found. You may not be authorized to view it.");

This presents an error message to the user and hides the current 5250 form, which can then only be interacted with by other script controlled actions.

Syntax

HIDE_CURRENT_FORM([sMessageText1] [, sMessageText2] ... [, sMessageTextN])

Parameters

sMessageText1	Optional. String that contains the first message to be issued.
sMessageText2 -> N	Optional. Strings that are to be concatenated with the previous message text (a separator space is added).

Return Value

None

Example

HIDE_CURRENT_FORM("Inventory item ", objListManager.AKey1[0], "was deleted.");

CURRENT_FORM Function

Gets the Form Name of the current RAMP screen.

Syntax

CURRENT_FORM()

Parameters

None

Return Value

String. Returns the Form name of the current screen, as a string:

Example

MyString = CURRENT_FORM() ;

SETCURSOR Function

Positions the cursor in a given row and column of the screen

Syntax

SETCURSOR([iRowNumber][,iColumnNumber][,iRowOffset])

Parameters

IRowNumber	Required. Integer that specifies the row number where to position the cursor.
IColumnNumber	Optional. Integer that specifies the column number where to position the cursor. Defaults to 1.
IRowOffset	Optional. Integer that specifies the row number of the first row in a browse list or subfile. This parameter only makes sense in situations where the cursor is to be positioned in a browse list or subfile row and the entry is selected pressing the Enter key. Note that in newlook, browse lists and subfiles are recognized as grids. In these grids, column headings are counted as rows. Hence the value of this parameter should equal to the row position of the topmost column heading literal.
	To find out the row position of the browselist or subfile, run the application in 5250 session until you reach the desired list. You should be able to see the row/column positions on the bottom right of the screen. Move the cursor until the column heading's top most piece of text.

Return Value

None.

Example

SETCURSOR(7, 10, 3);

ALERT_MESSAGE Function

Issue a message as an alert.

Syntax

ALERT_MESSAGE(sMessageText1 [, sMessageText2] ...)

Parameters

sMessageText1	Required. String that contains the first message to be issued.
sMessageText2	Optional. Other strings that are to be concatenated with the first message string (a separator space is automatically added between each string).

Return Value

None

Example

ALERT_MESSAGE("Inventory item ", objListManager.AKey1[0], "was deleted.");

CLEAR_MESSAGES Function

Clears all messages currently in the stack.

Syntax

CLEAR_MESSAGES()

Parameters

None

Return Value

None

Example

CLEAR_MESSAGES();

FATAL_MESSAGE Function

Issues a fatal message and causes the entire VLF application to terminate (unless it is being executed in design mode).

In design mode the message details are presented in the center of the RAMP panel area and the application continues to execute. In execution mode the entire VLF application terminates.

Syntax

FATAL_MESSAGE(sMessageText1 [, sMessageText2] [, sMessageText3]...)

Parameters

sMessageText1	Optional. String that contains the first message to be issued.
sMessageText2	Optional. Other strings that are to be concatenated with the first message string (a separator space is automatically added between each string).

Return Value

None

Example

FATAL_MESSAGE("Inventory item ", objListManager.AKey1[0] , "was deleted.");

MESSAGE Function

Issue a message.

Syntax

MESSAGE(sMessageText1 [, sMessageText2] ...)

Parameters

sMessageText1	Required. String that contains the first message to be issued.
sMessageText2	Optional. Other strings that are to be concatenated with the first message string (a separator space is automatically added between each string).

Return Value

None

Example

MESSAGE("Inventory item ", objListManager.AKey1[0] , "was deleted.");

AVSIGNALEVENT Function

Signal an event to the Framework filters and RAMP screens.

Syntax

```
AVSIGNALEVENT([sId] [,sTo] [,sAInfo1] [,sAInfo2] [,sAInfo3] [,sAInfo4] [,sAInfo5] [,nNInfo1] [,nNInfo2] [,nAInfo3] [,nNInfo4] [,nNInfo5])
```

Parameters

SId	Required.String containing an identifier of the Event.
STo	 Valid values are: FRAMEWORK = The signal is broadcast to the whole framework BUSINESSOBJECT = The signal is only broadcast to filters and RAMP screens in the current business object
sAInfo1	Optional. String containing additional information that the object listening for the signal can use.
sAInfo2	Optional. String containing additional information that the object listening for the signal can use.
sAInfo3	Optional. String containing additional information that the object listening for the signal can use.
sAInfo4	Optional. String containing additional information that the object listening for the signal can use.
sAInfo5	Optional. String containing additional information that the object listening for the signal can use.
nNInfo1	Optional. Number containing additional information that listening object may use.
nNInfo2	Optional. Number containing additional information that listening object may use.
nNInfo3	Optional. Number containing additional information

	that listening object may use.
nNInfo4	Optional. Number containing additional information that listening object may use.
nNInfo5	Optional. Number containing additional information that listening object may use.

Return Value

None

Example

This example signals that an entry has been deleted in the instance list:

AVSIGNALEVENT("Delete_List_Entry", "BUSINESSOBJECT", objListManager.AKey1[0]);

Also see Updating the Instance List from RAMP screens.

TRACE Function

Allows the user to add run time information from the script to the application trace panel.

Syntax

TRACE(sTraceText1 [, sTraceText2] ...)

Parameters

sTraceText1	Required.String that contains the trace information to be shown.
sTraceText2	Optional. String that is concatenated with the previous trace text (a separator space is added).

Return Value

None

Example

TRACE("Inventory item ", objListManager.AKey1[0] , "was deleted.");

RUNMACRO Function

Runs a newlook Macro.

Syntax

RUNMACRO(sMacroName)

Parameters

sMacroName	Required.String that contains the name of the newlook	
	Macro.	

Return Value

None

Example

RUNMACRO("MyMacro");

NAVIGATE_TO_JUNCTION Function

Navigates RAMP to a Junction.

Syntax

NAVIGATE_TO_JUNCTION(sJunctionName)

Parameters

SjunctionName	Required.String that contains the form name of the Junction.

Return Value

None

Example

/* Navigate to the nearest access junction */
NAVIGATE_TO_JUNCTION("uItemMasterBrowse");

Note that this function should only be invoked from an invoke script.

HANDLE_PROMPT Function

Causes an associated prompter form (VL Handler) to appear next to a field. The fields and the prompter forms are specified in the Special Field Handling area as described in Advanced Prompting.

Optionally additional information can be passed to or retrieved from the prompter form.

Syntax

```
HANDLE_PROMPT(sArgument1 [, sArgument2] [, sArgument3]...)
```

Parameters

SArgumentn Optional. String that contains any value the user defined prompter has bi-directional access to all nam fields in the 5250 screen.

Return Value

None

Example

```
if ( HANDLE_PROMPT() ) return;
```

Accessing the values passed as sArgument1, sArgument2, etc., in the prompter form

A function like this in a RAMP script: HANDLE_PROMPT("HELLO","THERE",123);

Is accessed like this in the prompter form: Invoke Method(#Com_Owner.uGet5250Field) Name(UARG1) Value(#Arg1Value) ... returns "HELLO" in #Arg1Value. Invoke Method(#Com_Owner.uGet5250Field) Name(UARG2) Value(#Arg2Value) ... returna "THERE" in #Arg2Value. Invoke Method(#Com_Owner.uGet5250Field) Name(UARG3) Value(#Arg3Value) ... returns "123" as a string in #Arg3Value. There is no limit on how many arguments you can pass.

Numeric values can be passed, but they will turn up as strings in the VL component, so they need to be converted back to a number again.

Referencing an un-passed argument does not cause a problem. This code:

```
#Arg15Value := "TEST"
```

```
Invoke Method(#Com_Owner.uGet5250Field) Name(UARG15) Value(#Arg15Value)
```

Would execute and leave #ARG15Value unchanged as "TEST", but you can actually tell whether the value was passed by doing this:

```
Invoke Method(#Com_Owner.uGet5250Field) Name(UARG15)
Value(#Arg15Value) Found(#Found)
```

If (#Found = TRUE) /* 15th argument was passed to HANDLE_PROMPT */ Else /* 15th argument was not passed)

As an example, you can use the additional arguments in a HANDLE_PROMPT function if you need access to values which are not on the screen from which the prompter form is invoked.

For instance, this could be used in a situation where customer information is entered on the first screen and an invoice number is prompted for on the second screen. If this invoice number is dependent on the customer information initially entered on the first screen and the information is not available to you on the second screen, you could store the required customer information in an objGlobal variable and pass it as HANDLE_PROMPT() parameters for proper select criteria in the prompter form code.

NAVIGATE_TO_DESTINATION Function

Navigates to a nominated 5250 Destination screen.

Note that if you specify the name of the current destination the request will be ignored. If you want to re run the script for a current destination use NAVIGATE_TO_PREV_DESTINATION(1).

Syntax

NAVIGATE_TO_DESTINATION(sDestinationName)

Parameters

Required. A string that contains the name of a Destination.
Destination.

Return Value

None

Example

NAVIGATE_TO_DESTINATION("Enrol Employee");

Note that this function should only be invoked from a button script.

NAVIGATE_TO_PREV_DESTINATION Function

Navigates the 5250 to a previously shown destination in this execution.

Syntax

NAVIGATE_TO_PREV_DESTINATION(iPreviousDestination)

Parameters

iPreviousDestination	Required. Integer that contains a number that indicates how many destinations backwards to navigate. The maximum allowed is 20.
	Note that previous destinations include every single destination that has been navigated through irrespective of whether it was shown or not. For example, you might execute a screen wrapper that passes through 2 destinations.

Return Value

None

Example

/* Re run INVOKE script to get the current 5250 destination screen*/
NAVIGATE_TO_PREV_DESTINATION(1);
/* Navigates to the previous 5250 destination screen*/
NAVIGATE_TO_PREV_DESTINATION(2);

Note that this function should only be invoked from a button script.

GET_MENU_OPTION_NUMBER Function

Searches the label fields on the current form looking for a menu option by name. If the menu option can be found an attempt is made to deduce an associated menu option number.

This function is useful in applications where the menu option number associated with an activity varies because it allows a text string that identifies the menu option in words to be dynamically converted to the associated menu option number.

This function uses a cache keyed by screen name/search text to optimize repeatedly performing the same operation.

The cache logic assumes that a menu option number on any given 5250 screen will not change within a signed-on session.

Syntax

GET_MENU_OPTION_NUMBER(sSearchText, bCaseInsensitive, bTryPrecdeingField)

L (
	sSearchText	String. Required. The menu option search string.	
	bCaseInsensitive	Boolean (true/false). Optional. Default is true. Indicates that the search should be case insensitive.	
	vTryPreceding	Boolean (true/false). Optional. Default is true. Indicates that when a label contains the search text, but a menu number cannot be deduced from it, that the preceding label field should be used as an alternate source for the menu number. This option accommodates menus where the menu option number and the menu text are in separate but adjacent label fields.	

Parameters

Return Value

String. The associated menu option number or an empty string if no menu option could be deduced.

Example

This code causes the message "Menu option number returned was 2" to be displayed:

/* Locate the menu option number of Office Tasks on the I5/OS Main menu */
var strMenuNumber = GET_MENU_OPTION_NUMBER("office tasks");
alert("Menu option number returned was " + strMenuNumber);

This code causes the message "Menu option number returned was 11" to be displayed:

/* Locate the menu option number of Client Access/400 tasks on the I5/OS Main menu */

var strMenuNumber = GET_MENU_OPTION_NUMBER("client acc"); alert("Menu option number returned was " + strMenuNumber);

STRIP_LEADING_NUMBERS Function

Returns the leading numbers from a string to the caller.

Syntax

STRIP_LEADING_NUMBERS(sSourceString)

Parameters

sSourceString	String. Required. The string from which
	the numbers are to be stripped.

Return Value

String. The stripped numbers.

Example

This code causes the message "String returned was 15" to be displayed: var strResult = STRIP_LEADING_NUMBERS("015. Office Tasks"); alert("String returned was " + strResult);

ADD_STRING Function

Defines a string by a unique number for use by other scripts. This function is especially useful in multilingual applications.

Syntax

ADD_STRING(iStringNumber , sText)

Parameters

iStringNumber	The number to be assigned to the string
sText	The string text

Return Value

None

Examples

See the STRING Function definition.

STRING Function

Returns the string for a given string identification number. This function is especially useful in multilingual applications.

Syntax

STRING(iStringNumber)

Parameters

iStringNumber The identification number of the string

Return Value

The string previously defined by ADD_STRING with the specified identification number or a string containing the text "String number n not found.".

Examples

If your sign-on function used the ADD_STRING() function to define multilingual strings like this based on different language codes:

```
ADD_STRING(1,"OK");
ADD_STRING(2,"Cancel");
ADD_STRING(3,"Customer not found");
```

Then all other scripts that needed to access a multi-lingual string would reference the function STRING(n) in their code in a language independent way. For example this code:

```
for (i = 0; i <= 4; i++)
{
alert( STRING(i) );
}
```

Would display the strings: String number 0 not found. OK Cancel Customer not found String number 4 not found Similarly, if your sign-on script had defined two strings like this:

ADD_STRING(1,"Customer number ");

ADD_STRING(2," could not be found or you are not authorized to view them.");

Then you could dynamically build a multi-lingual message in another script like this:

var strMessage = STRING(1) + CustomerNumber.toString() + STRING(2);
alert(strMessage);

OVERRIDE_BUTTONS_UNDEFINED_SCREENS Function

Applicable to Windows only.

Causes all undefined screens to show the 5250 function key bar.

Syntax

OVERRIDE_BUTTONS_UNDEFINED_SCREENS ()

Parameters

None.

Return Value

None

Remarks

This function turns on the function key bar for all screens in a session, but because function key bars should not appear in *defined* screens it must be hidden.

To use this special function you must do this:

- Set the Bottom Mask Height property for the *session* to 28. All destinations will inherit this mask setting by default.
- Use the OVERRIDE_BUTTONS_UNDEFINED_SCREENS() function in your sign-on script.

Once the option is turned on it will remain so for the duration of the session.

To show the function key bar for a destination screen, reset the Bottom Mask Height of the destination screen to zero. However, we do not recommend showing the function key bar for destination screens.

Note: when the undefined screen is a pop-up, the screen in the background (even though defined) might also show the function key bar although it will not be usable.

Examples

OVERRIDE_BUTTONS_UNDEFINED_SCREENS()

OVERRIDE_KEY_CAPTION_SCREEN Function

Assigns a new caption for a function key on a particular screen.

Syntax

OVERRIDE_KEY_CAPTION_SCREEN (sDestinationName,sKeyName,sOverrideCaption)

Parameters

sDestinationName	Required. A string that contains the name of a Destination.
sKeyName	Required. String that contains the name of the key. See Function Key Names for SENDKEY Function.
sOverrideCaption	Required. The new caption that will be used for the button

Return Value

None

Example

OVERRIDE_KEY_CAPTION_SCREEN("uDisplayEmployee", KeyF1, "Aide");

Notes

This function is very sensitive to where in a RAMP script it is used. If it is used in an INVOKE script for a destination, it should be placed just before the destination screen appears.

This function can also be used in a sign-on script.

OVERRIDE_KEY_CAPTION_ALL Function

Assigns a new caption for a function key on any screen.

Syntax

OVERRIDE_KEY_CAPTION_ALL (sKeyName,sOverrideCaption)

Parameters

sKeyName	Required. String that contains the name of the key. See Function Key Names for SENDKEY Function.
sOverrideCaption	Required. The new caption that will be used for the button

Return Value

None

Example

OVERRIDE_KEY_CAPTION_ALL(KeyF1, "Aide");

Notes

This function is usually used in a sign-on script. It can be used for multilingual applications to set all function key captions to another language.

AVSAVEVALUE Function

Saves an alphanumeric or numeric value onto the VLF virtual clipboard.

Syntax

AVSAVEVALUE(vValue, sID1, sID2, sID3, iInstance, sLanguage, bPersist)

Parameters

vValue	Required. Alphanumeric or numeric value to save to the virtual clipboard.		
	If this parameter is a JavaScript variable of type string, then the value is posed to the clipboard as an alphanumeric value and can therefore can only be sensibly be retrieved using the AVRESTORE <u>A</u> VALUE function (or equivalent).		
	If it is of type number it is posted as type numeric to the clipboard and can only be sensibly retrieved using the AVRESTORE <u>NVALUE</u> function (or equivalent).		
sID1	Required. String that contains the Virtual Clipboard identifier 1.		
sID2	Optional. String that contains the Virtual Clipboard identifier 2.		
sID3	Optional. String that contains the Virtual Clipboard identifier 3.		
iInstance	Optional. Integer that contains the instance number. Defaults to 1 when not specified. Instances are typically used to create lists of clipboard values and usually accompanied by another clipboard value that indicates how many entries currently exist in the list.		
sLanguage	age Optional. String that contains the language		

	code. Defaults to ALL languages when not specified.
bPersist	Optional. Boolean value that indicates whether or not a saved value should persist beyond the current execution of the RAMP application. Defaults to true. This parameter has no meaning for VLF-WEB RAMP applications because VLF virtual clipboard values never persist in WEB applications.

Return Value

None

Remarks

- Use AVSAVEVALUE in your RAMP scripts to save value in the VLF virtual clipboard. More information about the Virtual Clipboard can be found in The Virtual Clipboard in the Framework guide.
- For information about the parameter lengths, please refer to VF_SAVEAVALUE and VF_SAVENVALUE.
- The posting of clipboard values from RAMP scripts is asynchronous. When you post values they are not physically processed onto the clipboard until your RAMP script completes execution and yields control back to the framework.
- The virtual clipboard is primarily designed to pass information between RAMP scripts and RDML(X) code executing in filters, command handlers, etc.
- The virtual clipboard is not primarily designed to pass information between RAMP scripts. The JavaScript objGlobal object is a more efficient way to pass information exclusively between RAMP scripts.
- When a RAMP script executing in a web browser application posts values onto the virtual clipboard, they need to be sent to the server for subsequent access by RDML(X) code executing in filters or command handlers (because they are executing on the server). This means that the volume of information you place onto the clipboard will impact the amount of information that needs to be transmitted between the client and the server.

Examples

RDMLX code in a filter or command handler to save/restore clipboard values:

* Save values onto the clipboard

Invoke #avFrameworkManager.avSaveValue WithID1(Test) WithID2(EMPNO) FromAValue(("A0090") Invoke #avFrameworkManager.avSaveValue WithID1(Test) WithID2(SURNAME) FromAValue("FRED") Invoke #avFrameworkManager.avSaveValue WithID1(Test) WithID2(GIVENAME) FromAValue("BLOGGS") Invoke #avFrameworkManager.avSaveValue WithID1(Test) WithID2(POSTCODE) FromNValue(2150) Invoke #avFrameworkManager.avSaveValue WithID1(Test) WithID2(SALARY) FromNValue(123456.78)

* Restore values from the clipboard

Invoke #avFrameworkManager.avRestoreValue WithID1(Test) WithID2(EMPNO) ToAValue(#EMPNO) UseAValueDefault("NA") Invoke #avFrameworkManager.avRestoreValue WithID1(Test) WithID2(SURNAME) ToAValue(#SURNAME) UseAValueDefault("NA") Invoke #avFrameworkManager.avRestoreValue WithID1(Test) WithID2(GIVENAME) ToAValue(#GIVENAME) UseAValueDefault("NA") Invoke #avFrameworkManager.avRestoreValue WithID1(Test) WithID2(POSTCODE) ToNValue(#PostCode) UseNValueDefault(0) Invoke #avFrameworkManager.avRestoreValue WithID1(Test) WithID2(SALARY) ToNValue(#Salary) UseNValueDefault(0)

RAMP JavaScript code to perform the equivalent operations:

/* Save values onto the clipboard – note POSTCODE and SALARY are numeric */

AVSAVEVALUE("A0090","TEST","EMPNO"); AVSAVEVALUE("FRED","TEST","SURNAME"); AVSAVEVALUE("BLOGGS","TEST","GIVENAME");

AVSAVEVALUE(2150,"TEST","POSTCODE"); AVSAVEVALUE(123456.78,"TEST","SALARY");

/* Restore values from the clipboard */

```
var vEMPNO = AVRESTOREAVALUE("NA","TEST","EMPNO");
var vSURNAME = AVRESTOREAVALUE("NA","TEST","SURNAME");
var vGIVENAME = AVRESTOREAVALUE("NA","TEST","GIVENAME");
var vPOSTCODE = AVRESTORENVALUE(0,"TEST","POSTCODE");
var vSALARY = AVRESTORENVALUE(0,"TEST","SALARY");
```

AVRESTOREAVALUE and AVRESTORENVALUE Function

Restore an alphanumeric or numeric value from the VLF virtual clipboard.

Syntax

AVRESTOREAVALUE/AVRESTORENVALUE(Default, sID1, sID2, sID3, iInstance, sLanguage)

Parameters

Default	Required. String/Number that contains the default value to return if the value is not found .			
sID1	Required. String that contains the Virtual Clipboard identifier 1.			
sID2	Optional. String that contains the Virtual Clipboard identifier 2.			
sID3	Optional. String that contains the Virtual Clipboard identifier 3.			
iInstance	Optional. Integer that contains the instance number. Defaults to 1 when not specified			
sLanguage	Optional. String that contains the language code. Defaults to ALL languages when not specified.			

Return Value

None

Remarks

Use AVRESTOREAVALUE/AVRESTORENVALUE in your RAMP scripts to restore a value from the VLF virtual clipboard. More information about the Virtual Clipboard can be found in The Virtual Clipboard in the Framework guide

For information about the parameter lengths, please refer to VF_RESTOREAVALUE and VF_RESTORENVALUE in the Framework guide.

Examples

var sSavedSurname = AVRESTOREAVALUE("Not Found", "NewEmployee",
"Surname", "", 1, FRA);
var sSavedPostcode = AVRESTOREAVALUE(9999, "NewEmployee",
"Postcode");

SET_LOCK_MESSAGE Function

Set the message to show when the Framework locks up.

Syntax

SET_LOCK_MESSAGE(sText)

Parameters

Setting by Name:

sText	Required. String that contains the text of the			
	message.			

Return Value

None

Remarks

Using SET_LOCK_MESSAGE overrides the default message shown by the Framework. This function can be invoked at anytime in any script. It's recommended that, when used, the user message is set at the very beginning of the sign-on script.

This will have no effect when executing RAMP in a browser because locking is disabled in such an environment.

Examples

SET_LOCK_MESSAGE("This is my own message text for when the framework locks up")

ADD_UNKNOWN_FORM_GUESS Function

Function keys to send when an unknown form appears during RAMP navigation. Only available in Windows.

Syntax

ADD_UNKNOWN_FORM_GUESS(sKeyName)

Parameters

SKeyName	Required.String that contains the name of the key. Se			
	Function Key Names for SENDKEY Function.			

Return Value

None

Remarks

Use this function call in the session's sign-on script.

When an unknown 5250 screen is encountered, the Framework goes into a locked state if the *Lock Framework when an unknown 5250 form is displayed* property is turned on. The user will not be able to move around within the Framework until they navigate to a defined 5250 screen.

The ADD_UNKNOWN_FORM_GUESS function can help to work around such situation by specifying function keys to send as the user tries to execute a different Framework action (for example click on a different Application or Business Object, Command, etc.) without having to navigate to a defined 5250 screen. Before getting into a locked state, the Framework will send the added keys in the sequence they were added.

For example, your RAMP application may have many undefined F4=Prompt pop-up windows that are all closed by using F12=Cancel. You can instruct RAMP that when an unknown screen is on display (for example an F4=Prompt window) it should first try F12 (to see if it can close the window) before displaying the lock message.

It's up to the unknown 5250 screen to support the usage of the sent function keys and to the screen arriving after sending the keys to be defined for this functionality to work. This responsibility is up to the designer.

Care should be taken when using this function as it applies generically to all undefined screens.

Examples

ADD_UNKNOWN_FORM_GUESS(KeyF3); ADD_UNKNOWN_FORM_GUESS(KeyF12);

FORCE_POPUP_REFRESH Function

We strongly recommend you do not to use this function because the default behavior of pop-up windows is correct in most situations. If you think you need to use this function please contact your support representative.

RAMP contains force-to-front logic. This logic is invoked automatically whenever a 5250 pop-up is encountered as an undefined screen or when a function key or button is used from a destination form.

This logic can be turned off using FORCE_POP_UP_REFRESH(False).

Syntax

FORCE_POPUP_REFRESH(boolean)

Parameters

boolean. One of the following possible values:

true	Default. RAMP will attempt to bring the hidden pop up to the foreground.		
false	Do nothing.		

Return Value

None

Remarks

Using FORCE_POPUP_REFRESH only impacts the enablement of the force-to-front logic, *not when it is used*.

Please refer to Handling Pop-Ups for more information about this option.

SET_HANDLER_CAPTION Function

Set the current command handler caption to a new value.

Syntax

SET_HANDLER_CAPTION(sCaption)

Parameters

Setting by Name:

sCaption	Required. String that contains the new caption for the
	current command handler.

Return Value

None.

Remarks

Using SET_HANDLER_CAPTION overrides the default command handler caption shown by the Framework. This function can be invoked at anytime in any script.

This function is available to be used in Windows and Web RAMP applications.

Examples

SET_HANDLER_CAPTION("New Command Handler Caption")

Framework Objects that Scripts Can Refer To

A number of RAMP provided JavaScript objects make standard information accessible to all scripts. For example the JavaScript object **objUser** publishes properties *Name* and *Password*.

This means that you can access and pass around the name and password of the current user in your scripts like this:

if (objUser.Name == "QSECOFR") alert("Your are signed on as the security
officer!");

Note that these names are CASE SENSITIVE. Be careful to use exactly the same case as shown when writing scripts.

- objGlobal
- objFramework
- objApplication
- objBusinessObject
- objCommand
- objListManager
- objUser
- SUBFILE_ACCESSOR Object

To find out how you can quickly enter these objects and their properties in your scripts, see Using the Scripting Pop-up Menu.

objGlobal

objGlobal can be used to store your own properties.

This can be useful if you need to store information from one script and use it later in another script.

The information could be field values from a screen that need to be referred to by a later script.

Or it could identify which path a script is on, so that when the same screen is used by two paths, the script can determine which path it is on.

Property	Туре	Description
< <any name="" property="">></any>	string	Any property you want to assign to

Example

Save the path the user is on, and the item the user is working with (On Screen 1).

/* Store the Item number that the user entered - this field has to be defined on this form in newlook*/

```
objGlobal.utxtItemNumber = GETVALUE("utxtItemNumber");
```

/* Store the action that is being performed (so that shared screens can know whether its an add or a copy) */

objGlobal.uLastAction = "COPY";

Remember the path the user is on, and the item the user is working with (On Screen 4).

```
/* Get the action that is being performed */
```

```
if (objGlobal.uLastAction == "COPY")
```

```
{
```

ALERT_MESSAGE("Inventory item ", objGlobal.utxtItemNumber, "was copied from ", objListManager.AKey1[0]);

}

else

{

```
ALERT_MESSAGE("Inventory item was added. ", objGlobal.utxtItemNumber , "has been saved.");
```

Note that objGlobal is global within a 5250 session. Each 5250 session has its

own unique instance of objGlobal. For more information refer to Using the objGlobal Object.

objFramework

objFramework contains read only properties that provide information about the current framework to your scripts:

Property	Туре	Description
uCaption	string	The caption of the current framework
ExecutionEnvironment	string	Identifies the execution environment as "WIN" or "WEB"
flagDesignMode	boolean	Identifies whether the Framework is executing in design mode. Boolean value containing true or false.
Language	string	Identifies the current LANSA language code (eg: "ENG", "FRA", etc)
Partition	string	Identifies the current LANSA partition (eg: "DEM", "SYS")
TraceMode	string	Identifies whether the Framework is executing in Trace mode as "TRUE" or "FALSE"

objApplication

objApplication contains read only properties that provide information about the current application to your scripts:

Property	Туре	Description
uCaption	string	The caption of the current application.
uUserObjectType	string	The User Object Name / Type of the current application.

objBusinessObject

objFramework contains read only properties that provide information about the current business object to your scripts:

Property	Туре	Description
uCaption	string	The caption of the current business object.
uUserObjectType	string	The User Object Name / Type of the current business object.

objCommand

objFramework contains read only properties that provide information about the current command to your scripts:

Property	Туре	Description
uCaption	string	The caption of the current command.
uUserObjectType	string	The User Object Name / Type of the current command.
uAlphaArg1	String	The optional alpha argument 1 of the current VLF command handler
uAlphaArg2	String	The optional alpha argument 2 of the current VLF command handler
uNumArg1	Integer	The optional numeric argument 1 of the current VLF command handler
uNumArg2	Integer	The optional numeric argument 2 of the current VLF command handler
uExecReason	String	The reason that the current command handler was executed. This string contains "EXECUTE" or "ACTIVATE" indicating why the current RAMP command was executed. The value "ACTIVATE" is only applicable to visible VLF-WIN applicable to visible VLF-WIN application scripts. In all other contexts, including screen wrappers, the value "EXECUTE" is always used.

objListManager

objListManager contains read only properties that provide information about the instance list to your scripts.

Array properties

Array entry [0] is the value for the current entry in the instance list. (the entry that has focus)

Array entries [1], [2], [3] ... are the values for the selected entries in the instance list

Property	Туре	Description
AKey1[0] - AKey5[0]	string	The 5 Alpha identifying key values of the current instance of the instance list
NKey1[0] - NKey5[0]	String	The 5 Numeric identifying key values of the current instance of the instance list
VisualId1[0]	String	Visual Identifier 1 of the current Instance List entry
VisualId2[0]	String	Visual Identifier 2 of the current Instance List entry
AColumn1[0] – AColumn10[0]	String	The 10 Alpha Additional Column values of the current instance list entry
NColumn1[0] – NColumn10[0]	String	The 10 Numeric Additional Column values of the current instance list entry

Single value properties

Property	Туре	Description
TotalSelected	integer	The number of selected entries in the instance list.

For information about how to use the list manager object, see how to Interacting with Instance Lists in Scripts.

objUser

objUser contains read only properties that provide information about the current user to your scripts:

Property	Туре	Description
Name	string	The profile of the current User.
Password	string	The password of the current User.

SUBFILE_ACCESSOR Object

The SUBFILE_ACCESSOR object may be used to access a 5250 subfile from a script.

See the movie Using Subfile Accessor - 5 minutes.

Properties

SelectionColumnOffset	The offset of the selection column when it is in a different row to the matched column. Default is 0 (zero), indicating it is on the same subfile row.
SelectionKey	The key to be sent to select an entry in the subfile. The default is KeyEnter. If you don't want any key to be pressed, set this value to null (no quotes).
KeyPageDown	The key to be used to page the subfile down. The default is KeyPageDown.
Scrollable	Indicates whether the subfile may be scrolled down. The default is true.
Trace	Indicates whether detailed trace information should be produced from the subfile when trace mode is on. The default is true.
SelectionByCursor	Used when selection is to be done by cursor location. Default is false. Typically SelectionByCursorOffset is also set when using this value.
	When using SelectionByCursor this value indicates the offset between the matching row in the datagrid control and the actual 5250 screen line number the cursor should be positioned to.
SelectionByCursorOffset	For example, if the first data line in 5250

	subfile (visualized as a datagrid) was on line 12 of the 5250 display, you would set this property to 11. This indicates that a match on datagrid row 3 (say) would map to real 5250 screen line 3 + 11 = 14.
EndofFileMarker	Set the EndofFileMarker property to the value used in the subfile/browselist marker when the end of file has been reached. It will be then used to determine the end of the subfile when trying to select an entry. Default is "Bottom". This property can be set in RAMP scripts like this; objAccessor.EndofFileMarker = "End"; This will be used to determine when scrolling is no longer required.
UseMarker	Set the UseMarker property to <i>false</i> when the subfile/browse list does not use markers. Note that you must set the EndofFileMarker property to all or part of the message that appears when scrolling past the end of the subfile/browselist so the Subfile Accessor can determine when the end of the subfile/browselist is reached. The default is <i>true</i> .

Methods

Used to dispose of a SUBFILE_ACCESSOR object

Dispose()	when you have completed using in a script
SetSearchColumnNumber(1,2,3,4);	Defines a search column, by column number, to be used when looking for an entry in the subfile. Arguments are: 1 – Column Number 2 – Value to Search for 3 – Search without regard to case. Optional. Default false. 4 – Search using "contains" matching rather than exact equality. Optional. Default false.
SetSearchColumnName(1,2,3,4)	Defines a search column, by column name, to be used when looking for an entry in the subfile. Arguments are: 1 – Column Name 2 – Value to Search for 3 – Search without regard to case. Optional. Default false. 4 – Search using "contains" matching rather than exact equality. Optional. Default false.
SelectSubFileEntry()	Searches the subfile using the column search and selection details provided and selects the required subfile entry.
TracePage()	Dumps the current subfile details to the trace (if trace is active). No arguments.

SetSelectionColumnName(1,2,3)	Defines the selection column, by column name, to be used when selecting an entry in the subfile. Arguments are: 1 – Column number 2 – Selection value to be used 3 – Unselection value. Optional. Default is that automatic unselection of columns is not performed.
SetSelectionColumnNumber(1,2,3)	Defines the selection column, by column number, to be used when selecting an entry in the subfile. Arguments are: 1 – Column number 2 – Selection value to be used 3 – Unselection value. Optional. Default is that automatic unselection of columns is not performed.

Use the right mouse when coding scripts to generate base SUBFILE_ACCESSOR code. See Using the Scripting Pop-up Menu.

Function Key Names for SENDKEY Function

This table shows the function key names you need to use in the SENDKEY function and the corresponding 5250 and Windows key names.

Note that the key names are case sensitive and you must enter them exactly as shown here in the SENDKEY function.

SENDKEY Name	Windows Keyboard	5250 Key action description	Button Text
KeyAttn	Esc	sys attn	"Attn";
KeyClear	Shift Enter	Field Exit	"Clear";
KeyEnter	Enter	Enter	"Enter";
KeyHelp	alt F1	help	"Help";
KeyPageDown	Page Down	Page Down	"Page Up";
KeyPageUp	Page Up	Page Up	"Page Down";
KeyPrint	ctrl Pause	host print	"Print";
KeyReset	ctrl	reset	"Reset";
KeySysReq	shift Esc	sys req	"Sys Req";
KeyTestReq	alt Pause	test req	"Test Req";
KeyF1	F1	F1	"F1";
KeyF2	F2	F2	"F2";
KeyF3	F3	F3	"F3";
KeyF4	F4	F4	"F4";
KeyF5	F5	F5	"F5";

KeyF6	F6	F6	"F6";
KeyF7	F7	F7	"F7";
KeyF8	F8	F8	"F8";
KeyF9	F9	F9	"F9";
KeyF10	F10	F10	"F10";
KeyF11	F11	F11	"F11";
KeyF12	F12	F12	"F12";
KeyF13	shift F1	F13	"F13";
KeyF14	shift F2	F14	"F14";
KeyF15	shift F3	F15	"F15";
KeyF16	shift F4	F16	"F16";
KeyF17	shift F5	F17	"F17";
KeyF18	shift F6	F18	"F18";
KeyF19	shift F7	F19	"F19";
KeyF20	shift F8	F20	"F20";
KeyF21	shift F9	F21	"F21";
KeyF22	shift F10	F22	"F22";
KeyF23	shift F11	F23	"F23";
KeyF24	shift F12	F24	"F24";
KeyPA1	Esc 1	program attention 1	"PA1";
KeyPA2	Esc 2	program attention 2	"PA2";
KeyPA3	Esc 3	program attention 3	"PA3";

Example

SENDKEY(KeyEnter);

User-defined script functions

You can define your own JavaScript functions to be used in different RAMP scripts by editing a special JavaScript file UF_SY120.JS in the partition execute directory and creating your own functions based on function UF_MY_FUNCTION.

Note that the file names start with "U", not with "V".

You can edit this file with any editor or use the button at the top of the RAMP tools window to edit via NOTEPAD.

Your RAMP scripts will then be able to use the JavaScript functions you define.

If you are using RAMP in a web browser application, you will need to save uf_sy120.js to your web server.

To do this, select *Current RAMP Design Details* or *Shipped system and demonstration objects* options when you are saving the Framework on the web server.

Switching Off Recursion Checking

Each time a RAMP script is executed, the Framework checks if the script has been called recursively and flags an error if it has.

However, situations may arise where a script may appear to be called recursively, for example if a special screen appears two or more times in succession. In these cases the GLOBAL_flagRecursionCheck property can be used to switch off the recursion checking and avoid applications ending in error. The property can be used in scripts in this way:

```
var flagSaveCheckState = GLOBAL_flagRecursionCheck;
```

GLOBAL_flagRecursionCheck = false;

```
SENDKEY(KeyEnter);
```

```
GLOBAL_flagRecursionCheck = flagSaveCheckState;
```

Saving and restoring the state like this, rather than simply setting the global property to TRUE or FALSE is the best solution because this is a recursive situation. The Framework will handle three or four levels of recursion (depending on script size and system resources available) if a special screen appears this many times. Only the top recursion level will finally set the GLOBAL_flagRecursionCheck property back to TRUE again.

Debugging

Debug and Diagnostics - 2.5 minutes Common Scripting Errors Tracing Using ALERT_MESSAGE in Your Scripts

Debug and Diagnostics - 2.5 minutes

•

Play Movie to learn how to debug your application or read the Movie Summary.

Movie Summary

For movie Debug and Diagnostics - 2.5 minutes.

Switch on Tracing

Tracing is the first thing you need to do when debugging. Inspect the trace and look for screens that have not been recognized or that have a blank name.

Add Alert statements

An easy way to debug scripts is to add <u>Alert()</u> statements to display values in a pop-up window at run-time.

Add Alert_Message functions

Similarly, you can use the ALERT_MESSAGE Function to display values in pop-up windows.

Add Trace functions

If you do not want to interrupt application execution, but instead record values in the trace, use the TRACE Function in your script.

Debug your filters

If you want to debug your filters, you can use the avRecordTrace method in your filter program:

Invoke avFrameworkManager.avRecordTrace Component(#Com_Owner) Event('Search Button click handler started')

For more information see **Basic Tracing Service**.

Click on the Show Current newlook Form button

If the Framework ends on a screen it does not expect to be on, and you get a blank screen with an error message Unable to display form.

Use the Probe Screen button in Design mode

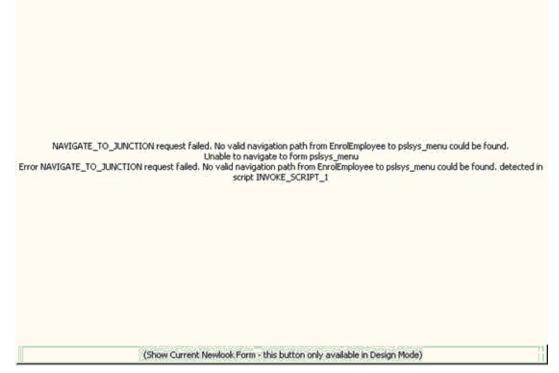
To find out what is know about the current screen.

Common Scripting Errors

NAVIGATE_TO_JUNCTION request failed Unable to display form Script with identifier XYZ not found Could not complete the operation due to error 80020101 Object expected Strange behavior in scripts Your script does not execute at all

NAVIGATE_TO_JUNCTION request failed

The execution of a RAMP screen results in a screen like this:



What does this error mean?

This error happens when the Framework has failed to build a navigation path because there is insufficient or no information for the navigation to complete successfully.

You can train the Framework to navigate from one RAMP screen to another until reaching the Destination screen. All Destinations screens need to have an INVOKE_SCRIPT_n associated with them. The first line in an INVOKE_SCRIPT_n looks like this:

```
/* Navigate to the nearest access junction */
NAVIGATE_TO_JUNCTION("<junction name>");
```

where <junction name> is the name given to a screen using newlook Designer and tagged as a Junction using the RAMP tools in the Framework.

Before starting the navigation, the Framework tries to create a navigation path to go from the screen currently showing to reach the <junction name> specified

in the NAVIGATE_TO_JUNCTION() function call.

The error means there is no valid path from the current screen to the <junction name>. The error shown In the example screen means there is no path to get from EnrolEmployee to a Junction named pslsys_menu (see the first message).

Solution

You need to manually execute the application starting at the screen that was showing in your runtime session, stop at every screen and verify that is has been defined, that the required scripts are there and that they are correct until you find the mistake. Somewhere along the navigation path you should find an undefined screen.

Do this:

Press the Show Current newlook Form button on the bottom of the error message screen to see the currently active screen. In the example the screen should be *EnrolEmployee* which is the name given to it using newlook Designer.

Choose the RAMP Tools option from the Framework menu and start a newlook session.

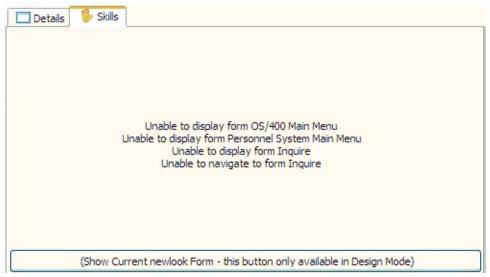
In newlook, display the screen that was showing in your runtime session. In our example, *EnrolEmployee*.

Once you have reached the screen causing the error (*EnrolEmployee*,), have a close look at the list of messages on the top right and answer the following questions:

Has the screen been defined in	If not, then it would not have any
the Framework as a Junction,	scripts and hence it would be unable
Destination or Special?	to navigate anywhere.
If the screen has been defined, have all the scripts been defined?	If all scripts have been defined, you need to review them.

Unable to display form

The execution of a RAMP screen results in a screen that looks like this:



What does this error mean?

The Framework has created a valid navigation path.

Most scripts check that the screen being shown is the one expected. That's why at the end of most scripts there is a line like this one:

```
/* Check for arrival at <form name> */
if ( !(CHECK_CURRENT_FORM("<form name>","Unable to display form
<form name>")) ) return;
```

The message Unable to display form suggests that at one stage during the navigation, a the identified screen was expected but another screen was received.

The message *Unable to navigate* is sent by the Destination's INVOKE_SCRIPT. It is a check to ensure that before running the Destination's script, the application is showing the proper screen. This avoids typing or sending key strokes in unwanted screens.

Sometimes you may not able to reach the undefined screen. This can happen when the screen which showed up unexpectedly was one that needs to be eliminated to allow the navigation to continue, typically a break message.

Solution

Press the Show Current newlook Form button to see the currently active screen. The screen shown is the unexpected one.

Select the RAMP Tools option in the Framework menu and manually perform the navigation that the RAMP screen was supposed to perform.

As you navigate through each one of the screens, answer the following questions:

Has the screen been defined?

Looking carefully at the scripts for the screen, does the script match what you do on the screen?

You should be able to manually reach the unexpected screen because you know what to do, what to type and what keys to press in each screen.

Script with identifier XYZ not found

The execution of a RAMP screen results in a screen like this:



What does this error mean?

This error happens when you delete a script and then execute the RAMP screen without saving and restarting the Framework.

Solution

Save and restart the Framework.

Could not complete the operation due to error 80020101

You execute one of your scripts and see an error message like this:

Could not complete the operation due to error 80020101. Error Could not complete the operation due to error 80020101. detected in script INVOKE_SCRIPT_1

What does this error mean?

Your script has a structural defect that prevents any attempt to execute it. For example, put this code:

```
if (1 == 2)
{
```

into a script and fail to add the required closing }. The RAMP editor will warn you about the missing }, but ignore the warning and go ahead and execute the script anyway. This will cause a 80020101 error because the script has a missing }.

The missing } means the whole script does not make any sense at all.

Similarly, this code causes an error because of the double closing square brackets:

```
SETVALUE("utxtBankAccountID",objListManager.AKey3[0]])
```

Solution

Look for "unbalanced" things in your script such as:

- An (without a closing/matching)
- An { without a closing/matching }
- An [without a closing/matching]
- A " or ' without a closing/match " or ' (an un-terminated string constant).
- An /* without a closing/matching */ (an un-terminated comment)

Other JavaScript constructs that are structurally incorrect.

Object expected

You execute one of your scripts an get an "Object Expected" error like this:

```
Object expected
Error Object expected detected in script INVOKE_SCRIPT_1
```

What does this error mean?

You have probably referred to something in your script that does not exist. The most common cause of this error is simple typographic errors or even case errors.

These script lines:

```
NaVIGATE_TO_JUNCTION("uOS400MainMenu");
NAVIGATE_TO_JUNCTIN("uOS400MainMenu");
```

will both produce an "object expected" error. The reason is that no object named NaVIGATE_TO_JUNCTION or NAVIGATE_TO_JUNCTIN actually exists. The correct JavaScript function name is NAVIGATE_TO_JUNCTION (remembering that JavaScript is case sensitive).

Solution

When you get an "Object expected Error" try:

- Checking the spelling of the name of object you are referencing.
- Checking the case of the name of the object you are referencing (eg: Userprofile or UserProfile).

Sometimes it is hard to tell exactly which line in your script is producing an error.

The easiest way to resolve this is to make liberal use of the JavaScript alert function. For example:

alert("About to navigate"); NaVIGATE_TO_JUNCTION("uOS400MainMenu"); alert("Navigation finished"); Would fairly quickly isolate that the NaVIGATE_TO_JUNCTION() line was the one causing the script failure.

Strange behavior in scripts

A very common cause of strange behavior in scripts comes from not using the "==" comparison correctly. This simple script demonstrates a very common and time wasting scripting problem:

```
var X = 1;
alert ("X is " + X);
if (X = 2)
{
    alert("X is 2");
  }
```

If you execute this script this first alert message will show X is 1 and the second will show shows X is 2 ... which is not possible.

The cause of this problem is of course that the if statement should have been

```
if (X == 2)
{
alert("X is 2");
}
```

Your script does not execute at all

Sometimes your script does not seem to execute at all.

Typically this is because it is because it is not being invoked in a 5250 screen navigation in the way that you thought it would be.

Use the *Framework*) -> (*Tracing*) -> Application Level menu options and trace the flow of control in your application to understand the navigation in detail. Generally this will reveal why your script is not being invoked.

Tracing

You can start tracing at any point in time during the execution of the Framework in design mode.

Use the Application Level trace facility to trace RAMP execution. To start tracing, click on the (Framework) menu, select (Tracing) -> Application Level.

Trace statements will appear in the Trace Window.

RAMP execution might produce a large number of statements. It will also produce long statements that will make it difficult to view in its entirety unless the window is enlarged.

Component	Event 🔨
VF_CH006	Script manager handling request ST
VF_CH006	Executing script INVOKE_SCRIPT
VF_CH006	Executing script NAVIGAT
VF_CH006	SETVALUE of userid to
VF_CH006	SETVALUE of userid co
VF_CH006	SETVALUE of psw to va
VF_CH006	SETVALUE of psw comp
VF_CH006	SENDKEY Enter request
VF_CH006	Screen named uLogin M
VF_CH006	Signal NL_SignOn is bei
VF_CH006	Signal NL_SignOn has b
VF_CH006	Session has now been
VF_CH006	Screen named uLogin M
VF_CH006	Executing script ELI
VF_CH006	SENDKEY Enter r
VF_CH006	Screen named u
VF_CH006	Screen named u
VF_CH006	Screen named u
VF_CH006	SENDKEY Enter c
VF_CH006	Execution of script E
VF_CH006	Screen named uOS400
VF_CH006	SENDKEY Enter complet
VF_CH006	Execution of script NAVIG
VF_CH006	Executing script NAVIGAT
VF_CH006	SETVALUE of Cmdline t
VF_CH006	SETVALUE of Cmdline c
VF_CH006	SENDKEY Enter request 💟
	Save Trace to File Clear Trace

For RAMP execution tracing, we recommend to use the Save Trace to File button to save the trace into a text file in your temp directory. The exact location and file name of the trace file produced will appear in a message.

Press the Messages button to find out about the location of the trace file.

Messages	
Trace saved in C:\DOCUME~1\PABLO~1.LAN\LOCALS~1\Temp\VF_SY001_APPTRACE.TXT	This is a general purpose "text" message see first level text for details
Close	

Adding Your Own Tracing Statements

The shipped Java Script function TRACE() allows you to add your own trace statements to the Application Level trace and the output of the trace statements is directed to the Application Level trace window.

For example, this trace statement:

```
TRACE("");
TRACE("Value of AKEY1 is =>" + objListManager.AKey1[0] + "<=");
TRACE("");</pre>
```

Generates this tracing:

Component	Event 🔨		
VF_CH006	Execution of script NAVIGATE_SCRIPT_7 - Na		
VF_CH006	Executing script NAVIGATE_SCRIPT_3 - Navi		
VF_CH006	SETVALUE of CmdIInE to value lansa run p		
VF_CH006	SETVALUE of CmdIInE completed		
VF_CH006	SENDKEY Enter requested.		
VF_CH006	Screen named psisys_menu has arrived an		
VF_CH006	Screen named pslsys_menu has identified		
VF_CH006	Screen named pslsys_menu is not a destin		
VF_CH006	SENDKEY Enter completed.		
VF_CH006			
VF_CH006	Value of AKEY1 is =>AUD<=		
VF_CH006			
VF_CH006	Execution of script NAVIGATE_SCRIPT_3 - Na		
VF_CH006	SETVALUE of Psisys_Opt to value 3 (type = stri 📃		
VF_CH006	SETVALUE of Psisys_Opt completed		
VF_CH006	SENDKEY Enter requested.		
VF_CH006	Screen named Type Empno has arrived and is be 🧮		
VF_CH006	Screen named Type Empno in a form not defined		
VF_CH006	Signal LockFramework is being queued.		
VF_CH006	Signal LockFramework has been queued.		
VF_CH006	Screen named Type Empno is not a destination d		
VE CHOOK	SENDKEY Enter completed		
	Save Trace to File Clear Trace		

The blank lines before and after the actual trace statement are generated by TRACE(""); simply to make it easier to read.

For more information about the trace statement refer to Script Functions.

Using ALERT_MESSAGE in Your Scripts

You might sometimes find that the easiest and quickest way to debug a problem is to put up a message box.

Using ALERT_MESSAGE() in your scripts causes a dialog box with a predefined message to appear.

ALERT_MESSAGE() can also display a mixture of text and variable values.

For example, if in one of your scripts you wanted to display the value of an Akey that is passed into the script, ALERT_MESSAGE() would look something like this:

ALERT_MESSAGE("The value of AKEY1 is =>" + objListManager.AKey1[0] + "<=");

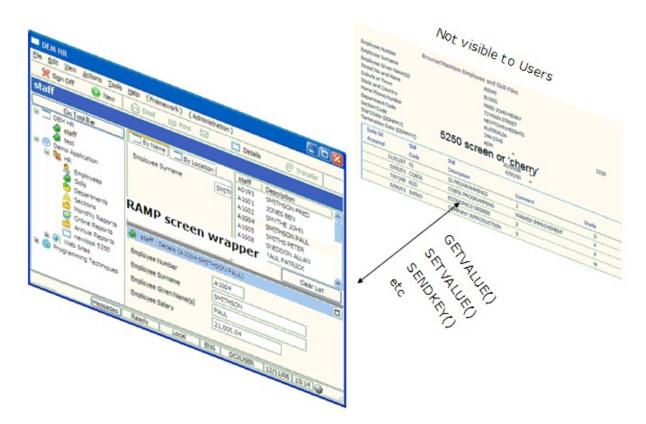
and during the execution a message box like this would be displayed:

Empno : Example 1 (A1008-ALLAN SNEDDON)	
EY1 is ==> AUD <==	
ОК	
ОК	

Screen Wrappers

RAMP screen wrappers are Visual LANSA components that access 5250 screens behind the scenes. The screens and fields accessed are defined in the usual manner by choreographing them.

A screen wrapper can pick values out of 5250 screens and present them to the user in completely different ways. Equally, a screen wrapper can accept input from the user and map it back into the 5250 screens to cause 5250 transactions to take place.



When to Use 5250 Screen Wrappers? Screen Wrapper Fundamentals Events Methods Examples

When to Use 5250 Screen Wrappers?

The main advantage of a screen wrapper is obvious. You can put a good looking, easy to use, high GUI veneer over 5250 screens, without having to spend the time and money required to analyze, rewrite and then retest all the business logic imbedded inside them as you would if you replaced them with VL components.

This is especially important for users to whom platform portability is of no real interest because they are content with a System i only solution.

Usage Examples

Some usage examples might include:

- A screen wrapper can pick values out of hidden 5250 screens and present it in completely different ways. For example, statistical information can be extracted and presented as a series of bar graphs (see Example 3: Show the System i Disk Usage).
- A screen wrapper can accept user input and then map it back into the 5250 screens so as to cause 5250 transactions to take place. For example, a VL component could allow high function, high volume order entry. When the user clicks Save, the order details are mapped into a series of 5250 screens and input.
- A screen wrapper might execute many 5250 screens from one click. For example a screen wrapper might display a list of 20 order numbers. When the user clicks OK all 20 orders are deleted by repeatedly executing a 5250 screen that only allows one order at a time to be deleted.

Role in Modernization Projects

For a customer happy with a System i dependent solution, a screen wrapper might be as far as they ever take application modernization.

Realistically, screen wrappers take time and money to develop, but probably significantly less than the equivalent VL component would, especially in the application testing phase of the modernization project.

Screen wrappers are not thrown away. When time and money permit, they may still be changed into proper VL components by removing their 5250 dependency.

You would expect modernization projects to go to market using a mix of 5250 screens, screen wrappers and VL components. For example, this might be the mix appropriate to an ISV:

- 85% 5250 screens to get to market ASAP.
- 10% screen wrappers to rapidly replace some heavily used and critical areas (eg: Order Entry) with something much better to use that adds a lot of business value.
- 5% VL components add high end value to the application (eg: E-Mail, PDF documents, MS-Excel spreadsheets, Web integration, etc).

Screen Wrapper Fundamentals

Define your screen wrapper

A screen wrapper is a VL reusable part of class VF_SY122. You must define it globally scoped as opposed to inside any type of routine.

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#PANL_1) Visible(False)

Key Points:

- Set the initial visibility to *False*. This will ensure it will never show up unless you want to. For example you might want to make it visible in design mode when a fatal error occurs to give you the option of seeing what the current 5250 screen is.
- You might want to make it a child of a panel attached to the center of the main panel. This will make it easier to see when you want to make it visible to track down fatal errors.

Set the uCommand property

In the command's *uInitialize* method routine, set the screen wrapper's *uCommand* property:

```
Mthroutine Name(uInitialize) Options(*REDEFINE)
* Do any initialization defined in the ancestor
Invoke Method(#Com_Ancestor.uInitialize)
Set Com(#myscreen_wrapper) Ucommand(#com_owner)
Endroutine
```

Key Points:

- Always set *uCommand* to #com_owner.
- Failure to set *uCommand* will result in an error message of type VF_INIT_ERROR.

Kick off execution by making RAMP available for a specific action

Usually you will invoke MakeRampAvailable Method inside the *uExecute* method of your command for the first time:

#myscreen_wrapper.MakeRampAvailable Foraction(Display)

Key Points:

• The first time you make RAMP available during the first execution of a

command it will take slightly longer for the event to be fired because RAMP is not connected to the host.

• The command regains control in the event routing handling RampAvailable Event.

Listen to the RampAvailable event

Once RAMP has connected and it's ready to be interacted with it will signal back to the command. It will pass the value of the action you requested. Typically this routine will consist of a CASE statement handling all the possible actions.

Listen to the RampMessage event

You write error handling logic and handle messages originating in your 5250 application in the RampMessage Event.

Events

RampMessage Event RampAvailable Event

RampMessage Event

A message is issued by RAMP or the underlying 5250 application.

Parameters

uMessageType	Char 256	String that specifies a type of message as per table below.
uMessageText	Char 132	String that contains the text of the message.

This table illustrates the available message types and their causes:

Туре	Cause	Comments
VF_ERROR	Fatal errors.	For whatever reason, RAMP has failed in the process of executing a request. For example, a failed navigation request.
VF_INFO	A message from the 5250 application.	Any message sent by the actual 5250 program running under the covers. For example, failed validation rules.
VF_INIT_ERROR	The Screen wrapper failed to initialize.	This usually happens when the session user object type supplied doesn't yield a defined session. Alternatively, if you haven't set the uCommand property (see Screen Wrapper Fundamentals).
VF_WAITCONNECTION	Issued every ½	Connections usually

	second while newlook is attempting a connection with the host.	complete very quickly. This type is provided only when for whatever reasons the connection is expected to take a little while.
VF_UNKNOWN_FORM	During navigation, an undefined form was detected.	

Remarks

It is entirely up to the developer how to handle different types of errors.

To cause a message to pop up automatically, use the

#com_owner.avshowmessages method. During development it might be useful to show the underlying newlook screen when a fatal error occurs. You can do so by changing the Screen wrapper's visibility and/or display position.

Example

Evtroutine Handling(#screen wrapper.uRampMessage) Umessagetype(#MsgType) Umessagetext(#MsgText) Case (#msgtype.value) When Value_Is('= VF_ERROR') * Optional. In design mode, making the screen wrapper visible allows you to show the 5250 screen.Set Com(#myscreen_wrapper) Visible(True) When Value_Is('= VF_INFO') Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value) When Value_Is('= VF_UNKNOWN_FORM') Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value) When Value_Is('= VF_INIT_ERROR') Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value) Endcase Endroutine

RampAvailable Event

RAMP has signaled it is interactive.

Parameters

ForAction	Char 256	String that specifies a user defined action identifier.
NextAction	Char 256	When a second action is attempted during the handling of an action, specify it here.

Remarks

Sometimes you might need to perform a second action within the same event handler.

For example, you make RAMP available for action A. For this action you navigate to a screen, then you get some values and depending on a condition you want to do action B or C, that is, navigate to a different screen.

Invoking the *MakeRampAvailable* method for action B or C while handling action A will cause a signal to the same event routine with undesirable consequences. It's only in these situations where you must set *NextAction*.

Example

```
Evtroutine Handling(#myscreen_wrapper.RampAvailable)
Foraction(#ForAction) Nextaction(#NextAction)
```

```
Case (#ForAction)

When Value_Is('= Display')

navigate to a screen

get value

If value is A

#NextAction := X

Else

Navigate to Y

endif

When Value_Is('= X')

When Value_Is('= Y')

Otherwise

Use Builtin(message_box_show) With_Args(ok ok info *component ('Unknown
```

ForAction>>' + #ForAction.Value + '<<')) Endcase

Endroutine

Also see MakeRampAvailable Method.

Methods

Screen wrappers drive the 5250 screens using using normal VL code methods supplied by component VF_SY122 (this is very similar to how the corresponding RAMP javascript functions work):

MakeRampAvailable Method

NavigateToScreen Method

SetValue Method

GetValue Method

SendKey Method

Current_Form Method

SetCursor Method

MakeRampAvailable Method

Make RAMP interactive for a specified action.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER)

#myscreen_wrapper.MakeRampAvailable Foraction(sAction)

Parameters

ForAction	Char 256 – Required	String that specifies an action. Actions are listened to in the <i>RampAvailable</i> event listener.
uUserObjectType	Char 32 - Optional	String that contains the user object type of the RAMP session specified when defining the session using the RAMP tools. When there is one session this parameter is not required.
uSession_Id	Char 40 - Optional	The session assigned to a destination. Defaults to *AUTO.

Return Value

None

Remarks

This method triggers the execution of a specific user-defined action. When you invoke this method the VLF will perform the connection if required. Once RAMP is available it will signal a *RampAvailable* event. The event routine listening to *RampAvailable* is where the main program logic is performed according to the specified action.

Examples

Invoke Method(#myscreen_wrapper.MakeRampAvailable) Foraction(Display) uSession_Id(SESSION_A)

Invoke Method(#myscreen_wrapper.MakeRampAvailable) Foraction(Update) uSession_Id(SESSION_A)

Invoke Method(#myscreen_wrapper.MakeRampAvailable) Foraction(Display) uUserObjectType(HumanResources)

Related Topic MakeRampAvailable Method.

NavigateToScreen Method

Navigate newlook to a screen.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)
Parent(#COM_OWNER)

#myscreen_wrapper.NavigateToScreen Name('EmpDetails') ReturnScreen

Parameters

Name	Char 256 –	String that contains the name of the
	Required	screen to navigate to.

Return Value

ReturnScreen Char 256 – Optional	String that contains the name of the screen wrapper 5250 screen when the navigation has completed.
-------------------------------------	--

Remarks

The screen to navigate can be a Junction or a Destination as defined in the choreographer.

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)
Parent(#COM_OWNER)
#myscreen_wrapper.Unavigatetoscreen) Name('EmpSkills')
Returnscreen(#vf_eltxtl)
```

SetValue Method

Set the content of a field on a 5250 screen to a value. The field may be identified by name or by its order on the screen.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER)

Setting by Name - #myscreen_wrapper.setvalue Infield(sField) Value(vValue)

Parameters

Setting by Name:

InField	Char 256 – Required	String that contains the RAMP field name.
Value	Variant – Required	String or number that contains the value to set the field to.

Setting by Order:

InField	Property - Required	<#myscreen_wrapper>.ByOrder_Field
Value	Variant – Required	String or number that contains the value to set the field to.
SpecialValue	Property – Optional	One of these two values: <#myscreen_wrapper>.First_Field – to set the value of the first field on the screen <#myscreen_wrapper>.Last_Field – to set the value of the last field on the screen
Order	Integer – Optional	The order of the field on the form starting from 1.

Return Value

None

Remarks

To set a value of a field on a screen by name, the field must be given a name in newlook Designer.

The use of field identification by order is more likely to be impacted by form layout changes than when using a name.

The initial setting of a field by order is more expensive to execute than by name, however screen field order details are cached so that the subsequent access is faster. The caching logic assumes that the relative order of a field on any particular screen will not change within a signed on 5250 session.

Examples

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER) Setting by Name - #myscreen_wrapper.setvalue Infield('uEmpno') Value(#EMPNO) Setting by Order - #myscreen_wrapper.setvalue Infield(#myscreen_wrapper.ByOrder_Field) Value(#Empno) Order(2)

GetValue Method

Get the value from a field on a RAMP screen.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)
Parent(#COM_OWNER)

#myscreen_wrapper.getvalue From(sField) Value(sValue)

Parameters

From	Char 256 – Required	String that contains the RAMP
		field name to get the value
		from.
		1

Return Value

Value	Variant –	Returns the value of the field as a
	Required	string or number.

Examples

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER) #myscreen_wrapper.getvalue From('uSurname') Value(#surname.value)

SendKey Method

Emulates the pressing of a function key.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)
Parent(#COM_OWNER)

#myscreen_wrapper.SendKey Key(#myscreen_wrapper.<key property>)

Parameters

Key	Property – Required	The property of #myscreen_wrapper that resolves to the desired key.
		For a list of these properties See the SENDKEY Names in <i>Function Key Names for</i> <i>SENDKEY Function</i> in lansa049.chm.

Return Value

Examples

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER) #myscreen_wrapper.Sendkey Key(#myscreen_wrapper.KeyEnter)

Current_Form Method

Gets the Form name of the current screen wrapper screen.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER)

#myscreen_wrapper.current_form Name(sName)

Parameters

None

Return Value

Name	Char 256 –	String that contains the name of the
	Required	current 5250 screen wrapper screen

Examples

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER) #myscreen_wrapper.current_form Name(#vf_eltxtl)

SetCursor Method

Positions the cursor in a given row and column of the screen. Optionally sends a key once the cursor has been positioned.

Syntax

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)
Parent(#COM_OWNER)

#myscreen_wrapper.SetCursor RowNum(iRowNum) ColNum(iColNum)
SendKey(#myscreen_wrapper.<key property>)

Parameters

RowNum	Integer – Required	Integer that specifies the row number where to position the cursor.
ColNum	Integer – Optional	Optional. Integer that specifies the column number where to position the cursor. Defaults to 1.
SendKey	Property - Optional	The property of #myscreen_wrapper that resolves to the desired key.
		For a list of these properties See the SENDKEY Names in <i>Function Key Names for</i> <i>SENDKEY Function</i> in lansa049.chm.

Return Value

None

Examples

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Parent(#COM_OWNER) #myscreen_wrapper.setcursor Rownum(10) Sendkey(#myscreen_wrapper.keyenter)

Examples

Example 1: Show Employee Details. Example 2: Show Employee Details and Skills Example 3: Show the System i Disk Usage

Example 1: Show Employee Details.

This example will navigate to the Browse and Maintain Employees screen which is part of the Personnel System.

👻 Basic details 🛛 🛷 Appro	we 🔲 Details 😒 Example 8 🤴 Skills	C Transfer	🚖 Email	🖙 Video	All Details
Employee Number	A0090				Save
Employee Surname	BLOGG				
Employee Given Name(s)	FRED JOHN HENRY				
Street No and Name	72 MAIN STREET				
Suburb or Town	NEWTOWN HEIGHTS				
State and Country	AUSTRALIA				
Post / Zip Code	2202				
Home Phone Number	344-2345				

To reach this screen, RAMP scripts will execute the following steps:

- Sign on
- Type *lansa run pslsys partition(dem)* in the command line and press Enter.
- Type 3 in the option field and press Enter.
- Type the employee number of the currently selected employee and press Enter.

• Press F21.

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #VF_AC010) Height(569) Layoutmanager(#MAIN_LAYOUT) Width(776) *

Group By Name(#XG HEAD) Fields(#EMPNO #SURNAME #GIVENAME #ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #PHONEHME **#DEPTMENT #SECTION)** * Body and Button arrangement panels Define_Com Class(#PRIM PANL) Name(#BUTTON PANEL) Displayposition(3) Height(569) Hint(*MTXTDF_DET1) Layoutmanager(#BUTTON_FLOW) Left(688) Parent(#COM_OWNER) Tabposition(3) Tabstop(False) Top(0) Width(88) Define_Com Class(#PRIM_PANL) Name(#BODY_HEAD) Displayposition(2) Height(569) Hint(*MTXTDF_DET1) Layoutmanager(#BODY HEAD FLOW) Left(0) Parent(#COM OWNER) Tabposition(2) Tabstop(False) Top(0) Verticalscroll(True) Width(688) * Attachment and flow layout managers Define_Com Class(#PRIM_ATLM) Name(#MAIN_LAYOUT) Define_Com Class(#PRIM_FWLM) Name(#BUTTON_FLOW) Direction(TopToBottom) Flowoperation(Center) Marginbottom(4) Marginleft(4) Marginright(4) Margintop(4) Spacing(4) Spacingitems(4) Define_Com Class(#PRIM_FWLM) Name(#BODY_HEAD_FLOW) Direction(TopToBottom) Marginbottom(4) Marginleft(4) Marginright(4) Margintop(4) Spacing(4) Spacingitems(4) Define_Com Class(#PRIM_FWLI) Name(#FWLI_EMPNO) Manage(#EMPNO) Parent(#BODY_HEAD_FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI SURNAME) Manage(#SURNAME) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI GIVENAME) Manage(#GIVENAME) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI ADDRESS1) Manage(#ADDRESS1) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI ADDRESS2) Manage(#ADDRESS2) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI ADDRESS3) Manage(#ADDRESS3) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI POSTCODE) Manage(#POSTCODE) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI PHONEHME) Manage(#PHONEHME) Parent(#BODY HEAD FLOW)

Define Com Class(#PRIM FWLI) Name(#FWLI SAVE BUTTON) Manage(#SAVE BUTTON) Parent(#BUTTON FLOW) * The save button Define Com Class(#PRIM PHBN) Name(#SAVE BUTTON) Caption(*MTXTDF_SAVE) Displayposition(1) Left(4) Parent(#BUTTON PANEL) Tabposition(1) Top(4) * Collection for detail fields Define_Com Class(#Prim_ACol<#prim_evef>) Name(#PanelFields) * Fields in the head area Define Com Class(#EMPNO.Visual) Displayposition(1) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(1) Top(4) Usepicklist(False) Width(209) Define_Com Class(#SURNAME.Visual) Displayposition(2) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(2) Top(27) Usepicklist(False) Width(324) Define Com Class(#GIVENAME.Visual) Displayposition(3) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(3) Top(50) Usepicklist(False) Width(324) Define_Com Class(#ADDRESS1.Visual) Displayposition(4) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(4) Top(73) Usepicklist(False) Width(363) Define_Com Class(#ADDRESS2.Visual) Displayposition(5) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(5) Top(96) Usepicklist(False) Width(363) Define_Com Class(#ADDRESS3.Visual) Displayposition(6) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY HEAD) Tabposition(6) Top(119) Usepicklist(False) Width(363) Define Com Class(#POSTCODE.Visual) Displayposition(7) Height(19) Hint(*MTXTDF DET1) Left(4) Parent(#BODY HEAD) Tabposition(7) Top(142) Usepicklist(False) Width(216) Define Com Class(#PHONEHME.Visual) Displayposition(8) Height(19) Hint(*MTXTDF DET1) Left(4) Parent(#BODY HEAD) Tabposition(8) Top(165) Usepicklist(False) Width(286) Define Com Class(#PRIM ATLM) Name(#ATLM 1) Define Com Class(#PRIM ATLI) Name(#ATLI 1) Attachment(Center) Parent(#ATLM 1) Define Com Class(#PRIM ATLI) Name(#ATLI 2) Attachment(Center) Manage(#BODY HEAD) Parent(#MAIN LAYOUT) Define Com Class(#PRIM ATLI) Name(#ATLI 3) Attachment(Right)

Manage(#BUTTON PANEL) Parent(#MAIN LAYOUT) Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Height(569) Parent(#COM OWNER) Visible(False) Width(688) Define Com Class(#PRIM ATLI) Name(#ATLI 4) Attachment(Center) Parent(#MAIN_LAYOUT) Define_Com Class(#PRIM_ATLI) Name(#ATLI_6) Attachment(Center) Manage(#myscreen_wrapper) Parent(#MAIN_LAYOUT) * _____ * Handle Initialization * _____ Mthroutine Name(uInitialize) Options(*REDEFINE) Define_Com Class(#Prim_evef) Name(#FormField) Reference(*dynamic) Invoke Method(#Com_Ancestor.uInitialize) For Each(#Control) In(#Body_Head.ComponentControls) If Ref Com(#Control) Is(*INSTANCE OF #prim evef) Set_Ref Com(#FormField) To(*dynamic #Control) Invoke Method(#PanelFields.Insert) Item(#FormField) Endif Endfor * Set the uCommand wrapper property. Set Com(#myscreen_wrapper) Ucommand(#com_owner) Endroutine * _____ * Handle Command Execution Mthroutine Name(uExecute) Options(*REDEFINE) Invoke Method(#Com Ancestor.uExecute) * The user has selected an Employee from the instance list. MakeRampAvailable will make sure the connection is in order and then signal back with the appropiate action Invoke Method(#myscreen wrapper.MakeRampAvailable) Foraction(ShowDetails) Set Com(#Save Button) Enabled(False) Endroutine * ______ * Event Handlers * RAMP has signalled it's available. What we do will depend on the #ForAction specified in the MakeRampAvailable method invocation. Evtroutine Handling(#myscreen_wrapper.RampAvailable) Foraction(#ForAction) Nextaction(#NextAction) Case (#ForAction) When Value_Is('= ShowDetails') * Navigate to a Destination that was previously named EmployeeDetailsAndSkills using newlook Designer. Use the ReturnScreen parameter to verify we are in the expected screen * once the navigation has completed Invoke Method(#myscreen_wrapper.navigatetoscreen) Name('EmployeeDetailsAndSkills') Returnscreen(#vf_eltxtl) * If the current screen is the expected one, get the values of the 5250 screen fields into the fields in this component If (#vf_eltxtl = 'EmployeeDetailsAndSkills') Invoke Method(#avListManager.GetCurrentInstance) Akey3(#EMPNO) #myscreen_wrapper.getvalue From('uSurname') Value(#surname.value) #myscreen_wrapper.getvalue From('uGivename') Value(#givename.value) #myscreen_wrapper.getvalue From('uAddress1') Value(#address1.value) #myscreen_wrapper.getvalue From('uAddress2') Value(#address2.value) #myscreen_wrapper.getvalue From('uAddress3') Value(#address3.value) #myscreen_wrapper.getvalue From('uHomePhone') Value(#phonehme.value) #myscreen_wrapper.getvalue From('uPostcode') Value(#POSTCODE.value) Endif When Value_Is('= UpdateDetails')

* Set the values of the fields in the newlook form with the ones from this component

#myscreen_wrapper.setvalue Infield('uSurname') Value(#surname.value) #myscreen_wrapper.setvalue Infield('uGivename') Value(#givename.value) #myscreen_wrapper.setvalue Infield('uAddress1') Value(#address1.value) #myscreen_wrapper.setvalue Infield('uAddress2') Value(#address2.value) #myscreen_wrapper.setvalue Infield('uAddress3') Value(#address3.value) #myscreen_wrapper.setvalue Infield('uHomePhone') Value(#phonehme.value) #myscreen_wrapper.setvalue Infield('uPostcode') Value(#POSTCODE.value) * Send the enter key to update the details in the 5250

#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyEnter)

Returnscreen(#vf_eltxtl)

Otherwise

Use Builtin(message_box_show) With_Args(ok ok info *component ('Unknown

```
ForAction>>' + #ForAction.Value + '<<'))</pre>
Endcase
Set Com(#myscreen_wrapper) Visible(False)
Endroutine
* Listen to messages from RAMP and the 5250 application
Evtroutine Handling(#myscreen_wrapper.RampMessage)
Umessagetype(#MsgType) Umessagetext(#MsgText)
Case (#msgtype.value)
When Value_Is('= VF_ERROR')
* Fatal messages reported by Ramp (e.g. Navigation request failed, etc). If in
design mode, show the underlying newlook screen. Otherwise, make the error
message
* appear in a message box on top of the command
If (#usystem.iDesignMode = true)
Set Com(#myscreen_wrapper) Visible(True)
Else
Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value)
#com_owner.avshowmessages
Endif
* Messages sent by the System i application or unknown form was encountered
When Value_Is('= VF_INFO' '= VF_UNKNOWN_FORM')
Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value)
* Failure to initialize RAMP. Could occur for mainly one of two reasons
When Value_Is('= VF_INIT_ERROR')
Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value)
#com owner.avshowmessages
When Value Is('= VF WAITCONNECTION')
Otherwise
Use Builtin(message box show) With Args(ok ok info *Component
('Unknown message type ' + #MsgType + 'encountered'))
Endcase
Endroutine
* Handle changes in any of the fields on the panel
* _____
Evtroutine Handling(#PanelFields<>.Changed)
* Enable the save button
Set Com(#SAVE BUTTON) Enabled(True)
* Lock the framework and set a message for the user
```

Use Builtin(bconcat) With_Args('Changes made to employee' #GiveName #Surname 'have not been saved yet.' 'Do you want to save them before continuing?') To_Get(#sysvar\$av) Set Com(#avFrameworkManager) Ulocked(USER) Ulockedmessage(#sysvar\$av) Endroutine * _____ _____ * Enter key pressed * _____ _____ Evtroutine Handling(#PanelFields<>.KeyPress) **Options(*NOCLEARMESSAGES *NOCLEARERRORS)** Keycode(#KeyCode) If Cond('#KeyCode.Value = Enter') * If there no changes have been made issue message and ignore enter If Cond('#SAVE BUTTON.Enabled *EQ True') Invoke Method(#Com Owner.Save) Else * Issue 'There are no changes to save' message Use Builtin(Message_box_show) With_Args(ok ok Info *Component *MTXTDF_NO_SAVE) Endif Endif Endroutine * _____ * Handle the save button * _____ _____ Evtroutine Handling(#SAVE BUTTON.Click) * Call the Save method Invoke Method(#Com Owner.Save) Endroutine * _____ * Handle Save * _____ _____ Mthroutine Name(Save) * Update data base Invoke Method(#myscreen wrapper.MakeRampAvailable) Foraction(UpdateDetails) Endroutine * _____

* Handle Termination
* -----

Mthroutine Name(uTerminate) Options(*REDEFINE) * Clean up the colelction of fields on the panel Invoke Method(#PanelFields.RemoveAll) * Do any termination defined in the ancestor Invoke Method(#Com_Ancestor.uTerminate) Endroutine End_Com

Example 2: Show Employee Details and Skills

This example is an extension of the previous one. It shows the same details but it also shows the skills in a Visual LANSA list view.

🔗 Basic details	Approv	/e 🔲 Details 😒 Example 8	🖖 Skills	C Transfer	📄 Email	🖙 Video	All Details
Employee Numbe	r	A0090					Save
Employee Surname		BLOGG					
Employee Given Name(s)		FRED JOHN HENRY					
Street No and Na	me	72 MAIN STREET					
Suburb or Town		NEWTOWN HEIGHTS					
State and Countr	y	AUSTRALIA					
Post / Zip Code		2202					
Home Phone Num	iber	344-2345					
Acquired	Skill Code	Skill	Commen	t	Gra		
21/01/87	CL	CL PROGRAMMING2	1		D		
0/00/01	COBOL	COBOL PROGRAMMING	MARKED	IMPROVE	P		
7/07/88	ECD	ECONOMICS DEGREE	2		P		
0/00/01	INTRO	COMPANY INTRODUCTION	3		M		
1/01/89	MANAGE1	MANAGEMENT COURSE 1	4		P		
25/09/88	MARKET2	MARKETING COURSE 2			F		
10/06/89	MARKET3	MARKETING COURSE 3			P		
2/08/89	MBA	MASTER BUSINESS			P		
1/01/92	RPG	RPG PROGRAMMING	6		P		

In this example you can see how to access a subfile/browselist:

Function Options(*DIRECT) Begin_Com Role(*EXTENDS #VF_AC010) Height(569) Layoutmanager(#MAIN_LAYOUT) Width(776) *

* Simple Field and Group Definitions

Group_By Name(#XG_HEAD) Fields(#EMPNO #SURNAME #GIVENAME #ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #PHONEHME #DEPTMENT #SECTION)

* Body and Button arrangement panels

Define Com Class(#PRIM PANL) Name(#BUTTON PANEL) Displayposition(2) Height(569) Hint(*MTXTDF_DET1) Layoutmanager(#BUTTON FLOW) Left(688) Parent(#COM OWNER) Tabposition(3) Tabstop(False) Top(0) Width(88) Define_Com Class(#PRIM_PANL) Name(#BODY_HEAD) Displayposition(1) Height(569) Hint(*MTXTDF_DET1) Layoutmanager(#BODY_HEAD_FLOW) Left(0) Parent(#COM_OWNER) Tabposition(2) Tabstop(False) Top(0) Verticalscroll(True) Width(688) * Attachment and flow layout managers Define_Com Class(#PRIM_ATLM) Name(#MAIN_LAYOUT) Define_Com Class(#PRIM_FWLM) Name(#BUTTON_FLOW) Direction(TopToBottom) Flowoperation(Center) Marginbottom(4) Marginleft(4) Marginright(4) Margintop(4) Spacing(4) Spacingitems(4) Define Com Class(#PRIM FWLM) Name(#BODY HEAD FLOW) Direction(TopToBottom) Marginbottom(4) Marginleft(4) Marginright(4) Margintop(4) Spacing(4) Spacingitems(4) Define_Com Class(#PRIM_FWLI) Name(#FWLI_EMPNO) Manage(#EMPNO) Parent(#BODY_HEAD_FLOW) Define_Com Class(#PRIM_FWLI) Name(#FWLI_SURNAME) Manage(#SURNAME) Parent(#BODY_HEAD_FLOW) Define_Com Class(#PRIM_FWLI) Name(#FWLI_GIVENAME) Manage(#GIVENAME) Parent(#BODY_HEAD_FLOW) Define_Com Class(#PRIM_FWLI) Name(#FWLI_ADDRESS1) Manage(#ADDRESS1) Parent(#BODY_HEAD_FLOW) Define_Com Class(#PRIM_FWLI) Name(#FWLI_ADDRESS2) Manage(#ADDRESS2) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI ADDRESS3) Manage(#ADDRESS3) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI POSTCODE) Manage(#POSTCODE) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI PHONEHME) Manage(#PHONEHME) Parent(#BODY HEAD FLOW) Define Com Class(#PRIM FWLI) Name(#FWLI SAVE BUTTON) Manage(#SAVE BUTTON) Parent(#BUTTON FLOW) * The save button Define Com Class(#PRIM PHBN) Name(#SAVE BUTTON) Caption(*MTXTDF SAVE) Displayposition(1) Left(4) Parent(#BUTTON PANEL) Tabposition(1) Top(4) * Collection for detail fields

Define_Com Class(#Prim_ACol<#prim_evef>) Name(#PanelFields)
* Fields in the head area

Define_Com Class(#EMPNO.Visual) Displayposition(1) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(1) Top(4) Usepicklist(False) Width(209)

Define_Com Class(#SURNAME.Visual) Displayposition(2) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(2) Top(27) Usepicklist(False) Width(324)

Define_Com Class(#GIVENAME.Visual) Displayposition(3) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(3) Top(50) Usepicklist(False) Width(324)

Define_Com Class(#ADDRESS1.Visual) Displayposition(4) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(4) Top(73) Usepicklist(False) Width(363)

Define_Com Class(#ADDRESS2.Visual) Displayposition(5) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(5) Top(96) Usepicklist(False) Width(363)

Define_Com Class(#ADDRESS3.Visual) Displayposition(6) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(6) Top(119) Usepicklist(False) Width(363)

Define_Com Class(#POSTCODE.Visual) Displayposition(7) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(7) Top(142) Usepicklist(False) Width(216)

Define_Com Class(#PHONEHME.Visual) Displayposition(8) Height(19) Hint(*MTXTDF_DET1) Left(4) Parent(#BODY_HEAD) Tabposition(8) Top(165) Usepicklist(False) Width(286)

Define_Com Class(#PRIM_ATLM) Name(#ATLM_1)

Define_Com Class(#PRIM_ATLI) Name(#ATLI_1) Attachment(Center) Parent(#ATLM_1)

Define_Com Class(#PRIM_ATLI) Name(#ATLI_2) Attachment(Center) Manage(#BODY_HEAD) Parent(#MAIN_LAYOUT)

Define_Com Class(#PRIM_ATLI) Name(#ATLI_3) Attachment(Right) Manage(#BUTTON_PANEL) Parent(#MAIN_LAYOUT)

Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Displayposition(3) Height(569) Parent(#COM_OWNER) Visible(False) Width(688)

Define_Com Class(#PRIM_ATLI) Name(#ATLI_4) Attachment(Center) Parent(#MAIN_LAYOUT)

Define_Com Class(#PRIM_ATLI) Name(#ATLI_6) Attachment(Center) Manage(#myscreen_wrapper) Parent(#MAIN_LAYOUT)

Define_Com Class(#PRIM_LTVW) Name(#skills) Componentversion(2) Displayposition(9) Fullrowselect(True) Height(229) Left(4) Parent(#BODY_HEAD) Showsortarrow(True) Tabposition(9) Top(188) Width(485) Define_Com Class(#PRIM_FWLI) Name(#FWLI_1) Manage(#skills) Parent(#BODY HEAD FLOW) Define_Com Class(#PRIM_LVCL) Name(#LVCL_2) Displayposition(2) Parent(#skills) Source(#SKILCODE) Width(17) Define_Com Class(#PRIM_LVCL) Name(#LVCL_3) Captiontype(ColumnHeadings) Displayposition(3) Parent(#skills) Source(#SKILDESC) Width(32) Define_Com Class(#PRIM_LVCL) Name(#LVCL_4) Captiontype(ColumnHeadings) Displayposition(4) Parent(#skills) Source(#COMMENT) Width(24) Define_Com Class(#PRIM_LVCL) Name(#LVCL_5) Displayposition(5) Parent(#skills) Source(#GRADE) Width(8) Widthtype(Characters) Define_Com Class(#PRIM_LVCL) Name(#LVCL_1) Caption('Acquired') Captiontype(Caption) Displayposition(1) Parent(#skills) Source(#VF ELTXTS) Width(18) Widthtype(Fixed) * _____ * Handle Initialization * _____ Mthroutine Name(uInitialize) Options(*REDEFINE) Define_Com Class(#Prim_evef) Name(#FormField) Reference(*dynamic) Invoke Method(#Com_Ancestor.uInitialize) For Each(#Control) In(#Body Head.ComponentControls) If_Ref Com(#Control) Is(*INSTANCE_OF #prim_evef) Set Ref Com(#FormField) To(*dynamic #Control) Invoke Method(#PanelFields.Insert) Item(#FormField) Endif Endfor * Set the uCommand wrapper property. Set Com(#myscreen wrapper) Ucommand(#com owner) Endroutine * _____ * Handle Command Execution * _____ Mthroutine Name(uExecute) Options(*REDEFINE) Invoke Method(#Com Ancestor.uExecute)

* The user has selected an Employee from the instance list. MakeRampAvailable will make sure the connection is in order and then signal back with the appropriate action Invoke Method(#myscreen_wrapper.MakeRampAvailable) Foraction(ShowDetails) Set Com(#Save_Button) Enabled(False) Endroutine _______ * Event Handlers * RAMP has signalled it's available. What we do will depend on the #ForAction specified in the MakeRampAvailable method invocation. Evtroutine Handling(#myscreen_wrapper.RampAvailable) Foraction(#ForAction) Nextaction(#NextAction) Case (#ForAction) When Value_Is('= ShowDetails') * Navigate to a Destination that was previously named EmployeeDetailsAndSkills using newlook Designer. Use the ReturnScreen parameter to verify we are in the expected screen * once the navigation has completed Invoke Method(#myscreen_wrapper.navigatetoscreen) Name('EmployeeDetailsAndSkills') Returnscreen(#vf_eltxtl) * If the current screen is the expected one, get the values of the 5250 screen fields into the fields in this component and the skills into the skills list view If (#vf_eltxtl = 'EmployeeDetailsAndSkills') #myscreen wrapper.sendkey(#myscreen wrapper.KeyF21) Invoke Method(#avListManager.GetCurrentInstance) Akey3(#EMPNO) #myscreen_wrapper.getvalue From('uSurname') Value(#surname.value) #myscreen_wrapper.getvalue From('uGivename') Value(#givename.value) #myscreen_wrapper.getvalue From('uAddress1') Value(#address1.value) #myscreen_wrapper.getvalue From('uAddress2') Value(#address2.value) #myscreen_wrapper.getvalue From('uAddress3') Value(#address3.value) #myscreen_wrapper.getvalue From('uHomePhone') Value(#phonehme.value) #myscreen_wrapper.getvalue From('uPostcode') Value(#POSTCODE.value) #com_owner.uGetSkills Gridname('uSkillsGrid') Endif When Value Is('= UpdateDetails')

#myscreen_wrapper.setvalue Infield('uSurname') Value(#surname.value) #myscreen_wrapper.setvalue Infield('uGivename') Value(#givename.value) #myscreen_wrapper.setvalue Infield('uAddress1') Value(#address1.value) #myscreen_wrapper.setvalue Infield('uAddress2') Value(#address2.value) #myscreen_wrapper.setvalue Infield('uAddress3') Value(#address3.value) #myscreen_wrapper.setvalue Infield('uHomePhone') Value(#phonehme.value) #myscreen_wrapper.setvalue Infield('uPostcode') Value(#POSTCODE.value) #myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyEnter) Returnscreen(#vf_eltxtl) * Update the instance list using the "quick update" method Use Builtin(BConcat) With_Args(#GiveName #SurName) To_Get(#FullName) Invoke Method(#avListManager.UpdateListEntryData) Akey1(#Deptment) Akey2(#Section) Akey3(#Empno) Visualid2(#FullName) Acolumn1(#Phonehme) Acolumn2(#Address1) Ncolumn1(#PostCode) Businessobjecttype(EMPLOYEES) * Disable the save button again Set Com(#SAVE_BUTTON) Enabled(False) * Drop the framework lock as no updates are outstanding now Set Com(#avFrameworkManager) Ulocked(FALSE) Otherwise Use Builtin(message_box_show) With_Args(ok ok info *component ('Unknown ForAction>>' + #ForAction.Value + '<<'))</pre> Endcase Set Com(#myscreen_wrapper) Visible(False) Endroutine * Traverse the skills subfile/browselist by column name Mthroutine Name(uGetSkills) Define Map For(*input) Class(#vf eltxtl) Name(#GridName) Define Map For(*input) Class(#vf eltxtl) Name(#nxtpage) Mandatory('+') Define Field(#colcount) Type(*dec) Length(2) Decimals(0) Define Field(#rowcount) Type(*dec) Length(4) Decimals(0) Define Field(#column) Type(*dec) Length(2) Decimals(0) Default(0) Define Field(#row) Type(*dec) Length(2) Decimals(0) Default(0) Define Field(#colname) Type(*char) Length(50) Define Field(#headrows) Type(*dec) Length(2) Decimals(0) Default(0) Clr List Named(#skills) Dowhile (#nxtpage *NE ") * Get the total number of subfile rows #myscreen wrapper.getvalue From(#GridName.value + ".RowCount")

Value(#rowcount) * Get the total number of subfile heading rows #myscreen_wrapper.getvalue From(#GridName.value + ".HeadRows") Value(#headrows) * Subtract one because the row collection is zero based. #rowcount -= 1 Begin_Loop Using(#row) From(#headrows) To(#rowcount) * get the number of subfile colums #myscreen_wrapper.getvalue From(#GridName.value + ".Columns.Count") Value(#colcount) Begin_Loop Using(#column) To(#colcount) * get the column name. Use a method to make the code easier to read #com_owner.uGetColName Ugridname(#GridName.value) Ucolnumber(#column) Ucolname(#colname) * for the appropiate column, get the cell value Case (#colname) When Value_Is(= 'DateSklAcquired') #com_owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#vf_eltxts) When Value_Is(= 'SkillCode') #com_owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#skilcode) When Value_Is(= 'SkillDescription') #com_owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#skildesc) When Value Is(= 'Comment') #com owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#comment) When Value Is(= 'Grade') #com owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#grade) Endcase End Loop * Sometimes newlook treats rows without data as valid rows so add only the ones where at least one field has data If Null (#skilcode #skildesc #comment #grade) Else Add_Entry To_List(#skills) Endif

End Loop * If there is another page, page down #myscreen_wrapper.getvalue From(#GridName.value + ".Marker") Value(#nxtpage.value) If (#nxtpage.value *NE ") #myscreen wrapper.sendkey Key(#myscreen wrapper.KeyPageDown) Endif Endwhile Endroutine Mthroutine Name(uGetColName) Define_Map For(*input) Class(#vf_eltxtl) Name(#uGridName) Define_Map For(*input) Class(#vf_elnum) Name(#uColNumber) Define_Map For(*output) Class(#vf_eltxtl) Name(#uColName) * The column collection is zero based but Begin Loop must start at minimum of 1. #ucolnumber -= 1 #myscreen_wrapper.getvalue From(#uGridName.value + ".Columns(" + #uColNumber.asstring + ").Name") Value(#ucolname.value) Endroutine Mthroutine Name(uGetCellValue) Define_Map For(*input) Class(#vf_eltxtl) Name(#uGridName) Define_Map For(*input) Class(#vf_elnum) Name(#uColNumber) Define_Map For(*input) Class(#vf_elnum) Name(#uRowNumber) Define_Map For(*output) Class(#vf_eltxtl) Name(#uCellvalue) * The column collection is zero based but Begin Loop must start at minimum of 1. #ucolnumber -= 1 #myscreen wrapper.getvalue From(#uGridName.value + ".Columns(" + #uColNumber.asstring + ").Cells(" + #uRowNumber.asstring + ").Text") Value(#ucellvalue.value) Endroutine * Listen to messages from RAMP and the 5250 application Evtroutine Handling(#myscreen wrapper.RampMessage) Umessagetype(#MsgType) Umessagetext(#MsgText) Case (#msgtype.value) When Value Is('= VF ERROR') * Fatal messages reported by Ramp (e.g. Navigation request failed, etc). If in design mode, show the underlying newlook screen. Otherwise, make the error

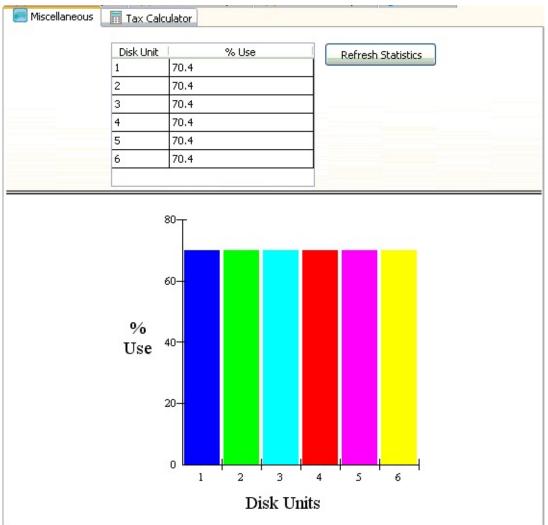
message

```
* appear in a message box on top of the command
If (#usystem.iDesignMode = true)
Set Com(#myscreen_wrapper) Visible(True)
Else
Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value)
#com_owner.avshowmessages
Endif
* Messages sent by the System i application or unknown form was encountered
When Value_Is('= VF_INFO' '= VF_UNKNOWN_FORM')
Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value)
* Failure to initialize RAMP. Could occur for mainly one of two reasons
When Value_Is('= VF_INIT_ERROR')
Message Msgid(dcm9899) Msgf(dc@m01) Msgdta(#msgtext.value)
#com_owner.avshowmessages
When Value_Is('= VF_WAITCONNECTION')
Otherwise
Use Builtin(message_box_show) With_Args(ok ok info *Component
('Unknown message type ' + #MsgType + 'encountered'))
Endcase
Endroutine
  _____
* Handle changes in any of the fields on the panel
* _____
Evtroutine Handling(#PanelFields<>.Changed)
* Enable the save button
Set Com(#SAVE BUTTON) Enabled(True)
* Lock the framework and set a message for the user
Use Builtin(bconcat) With Args('Changes made to employee' #GiveName
#Surname 'have not been saved yet.' 'Do you want to save them before
continuing?') To Get(#sysvar$av)
Set Com(#avFrameworkManager) Ulocked(USER)
Ulockedmessage(#sysvar$av)
Endroutine
* _____
* Enter key pressed
* _____
Evtroutine Handling(#PanelFields<>.KeyPress)
Options(*NOCLEARMESSAGES *NOCLEARERRORS)
Keycode(#KeyCode)
```

If Cond('#KeyCode.Value = Enter')
* If there no changes have been made issue message and ignore enter
If Cond('#SAVE_BUTTON.Enabled *EQ True')
Invoke Method(#Com_Owner.Save)
Else
* Issue 'There are no changes to save' message
Use Builtin(Message_box_show) With_Args(ok ok Info *Component
*MTXTDF_NO_SAVE)
Endif
Endif
Endroutine
*
* Handle the save button *
Evtroutine Handling(#SAVE_BUTTON.Click)
* Call the Save method
Invoke Method(#Com_Owner.Save)
Endroutine
*
* Handle Save
*
Mthroutine Name(Save)
* Update data base
Invoke Method(#myscreen_wrapper.MakeRampAvailable)
Foraction(UpdateDetails)
* If update completed okay
Endroutine
** Handle Termination
Mthroutine Name(uTerminate) Options(*REDEFINE)
* Clean up the colection of fields on the panel
Invoke Method(#PanelFields.RemoveAll)
* Do any termination defined in the ancestor
Invoke Method(#Com_Ancestor.uTerminate) Endroutine
End_Com

Example 3: Show the System i Disk Usage

A screen wrapper can pick values out of hidden 5250 screen(s) and present it in completely different ways. This example shows the disk usage of a System i graphically:



To access the work with disk status screen type *wrkdsksts* in the command line. The name given to the Work with Disk Status screen in this example is "DiskStatus".

When in the disk status screen, read the %Use column of the subfile and feed the data to the graph.

* * * COMPONENT: STD_PANL * *

Function Options(*DIRECT) Begin Com Role(*EXTENDS #VF AC010) Height(559) Hint(*MTXTDF_DET1) Layoutmanager(#ATLM_1) Width(557) Define Com Class(#PRIM GRID) Name(#DiskSts) Displayposition(1) Height(150) Left(109) Parent(#PANL_2) Rowheight(19) Tabposition(1) Top(15) Width(212) Define_Com Class(#PRIM_GDCL) Name(#GDCL_1) Caption('Disk Unit') Captiontype(Caption) Displayposition(1) Parent(#DiskSts) Source(#VF_ELTYPE) Width(29) Define_Com Class(#PRIM_GDCL) Name(#GDCL_2) Caption('% Use') Captiontype(Caption) Displayposition(2) Parent(#DiskSts) Readonly(False) Source(#VF_ELTXTS) Width(30) Widthtype(Remainder) Define Com Class(#PRIM GRPH) Name(#GRPH 1) Displayposition(1) Height(370) Left(0) Parent(#PANL_3) Scatterstyle(SymbolAtPoints+Solid) Surfacestyle(ConnectLinesInBlack) Tabposition(1) Top(0) Width(557) Xcaption('Disk Units') Ycaption('% Use') Define_Com Class(#PRIM_GRCL) Name(#GRCL_1) Columnrole(Label) Displayposition(1) Parent(#GRPH_1) Source(#VF_ELTYPE) Define_Com Class(#PRIM_GRCL) Name(#GRCL_2) Columnsymbol(HollowUpTriangle) Displayposition(2) Parent(#GRPH_1) Source(#VF_ELWIDP) Define_Com Class(#vf_sy122) Name(#myscreen_wrapper) Displayposition(3) Height(513) Left(144) Parent(#PANL_1) Top(24) Visible(False) Width(593) Define Com Class(#PRIM_PANL) Name(#PANL_1) Displayposition(1) Height(559) Layoutmanager(#SPLM 1) Left(0) Parent(#COM OWNER) Tabposition(1) Tabstop(False) Top(0) Width(557) Define Com Class(#PRIM ATLM) Name(#ATLM 1) Define Com Class(#PRIM ATLI) Name(#ATLI 1) Attachment(Center) Manage(#PANL 1) Parent(#ATLM 1) Define Com Class(#PRIM SPLM) Name(#SPLM 1) Define Com Class(#PRIM PANL) Name(#PANL 2) Displayposition(1) Height(185) Layoutmanager(#FWLM 1) Left(0) Parent(#PANL 1) Tabposition(2) Tabstop(False) Top(0) Width(557) Define Com Class(#PRIM PANL) Name(#PANL 3) Displayposition(2) Height(370) Layoutmanager(#ATLM_2) Left(0) Parent(#PANL_1) Tabposition(3) Tabstop(False) Top(189) Width(557) Define Com Class(#PRIM SPLI) Name(#SPLI 1) Manage(#PANL 2) Parent(#SPLM 1) Weight(1)

Define_Com Class(#PRIM_SPLI) Name(#SPLI_2) Manage(#PANL_3) Parent(#SPLM 1) Define_Com Class(#PRIM_ATLM) Name(#ATLM_2) Define Com Class(#PRIM ATLI) Name(#ATLI 2) Attachment(Center) Manage(#GRPH_1) Parent(#ATLM_2) Define_Com Class(#PRIM_FWLM) Name(#FWLM_1) Direction(TopToBottom) Flowoperation(Center) Margintop(15) Spacingitems(2) Define_Com Class(#PRIM_FWLI) Name(#FWLI_3) Manage(#DiskSts) Parent(#FWLM_1) Define_Com Class(#PRIM_PHBN) Name(#PHBN_1) Caption('Refresh Statistics') Displayposition(2) Left(331) Parent(#PANL_2) Tabposition(2) Top(15) Width(117) Define_Com Class(#PRIM_FWLI) Name(#FWLI_6) Manage(#PHBN_1) Parent(#FWLM 1) Mthroutine Name(uInitialize) Options(*REDEFINE) * Do any initialization defined in the ancestor Invoke Method(#Com_Ancestor.uInitialize) Set Com(#grph_1) Graphtype(Bar) Set Com(#myscreen_wrapper) Ucommand(#com_owner) Endroutine Mthroutine Name(uExecute) Options(*REDEFINE) * Do any execution logic defined in the ancestor Invoke Method(#Com_Ancestor.uExecute) Invoke Method(#myscreen_wrapper.MakeRampAvailable) Foraction(ShowDiskStatus) Endroutine Evtroutine Handling(#myscreen wrapper.RampAvailable) Foraction(#ForAction) * Get the employee number of the employee whose details are to be displayed. Case (#ForAction) When Value Is('= ShowDiskStatus') #myscreen wrapper.current form Name(#vf eltxtl) If (#vf eltxtl *NE 'DiskStatus') Invoke Method(#myscreen wrapper.navigatetoscreen) Name('DiskStatus') Returnscreen(#vf eltxtl) Endif If (#vf_eltxtl = 'DiskStatus') #myscreen wrapper.sendkey Key(#myscreen wrapper.KeyF10)

#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyF5) #com owner.uGetDiskStatus Gridname('uDiskStatus') Endif Otherwise Use Builtin(message_box_show) With_Args(ok ok info *component ('Incorrect ForAction>>' + #ForAction.Value + '<<')) Endcase Endroutine Mthroutine Name(uGetDiskStatus) Define_Map For(*input) Class(#vf_eltxtl) Name(#GridName) Define_Map For(*input) Class(#vf_eltxtl) Name(#nxtpage) Mandatory('+') Define Field(#colcount) Type(*dec) Length(2) Decimals(0) Define Field(#rowcount) Type(*dec) Length(4) Decimals(0) Define Field(#column) Type(*dec) Length(2) Decimals(0) Default(0) Define Field(#row) Type(*dec) Length(2) Decimals(0) Default(0) Define Field(#colname) Type(*char) Length(50) Define Field(#headrows) Type(*dec) Length(2) Decimals(0) Default(0) Clr List Named(#DiskSts) Clr_List Named(#grph_1) Dowhile ((#nxtpage *NE ") And (#nxtpage *NE 'Bottom')) #myscreen_wrapper.getvalue From(#GridName.value + ".RowCount") Value(#rowcount) #myscreen_wrapper.getvalue From(#GridName.value + ".HeadRows") Value(#headrows) * Subtract one because the row collection is zero based. #rowcount -= 1 Begin Loop Using(#row) From(#headrows) To(#rowcount) #myscreen wrapper.getvalue From(#GridName.value + ".Columns.Count") Value(#colcount) Begin Loop Using(#column) To(#colcount) #com owner.uGetColName Ugridname(#GridName.value) Ucolnumber(#column) Ucolname(#colname) Case (#colname) When Value Is(= 'Unit') #com owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#vf_eltype) When Value Is(= 'Used') #com owner.uGetCellValue Ugridname(#GridName.value) Ucolnumber(#column) Urownumber(#row) Ucellvalue(#vf eltxts)

```
#VF ELWIDP := #vf eltxts.trim.asnumber
Endcase
End Loop
Add Entry To List(#DiskSts)
Add_Entry To_List(#grph_1)
End Loop
#myscreen_wrapper.getvalue From(#GridName.value + ".Marker")
Value(#nxtpage.value)
If ((#nxtpage.value *NE ") And (#nxtpage.value *NE 'Bottom'))
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyPageDown)
Endif
Endwhile
Endroutine
Mthroutine Name(uGetColName)
Define_Map For(*input) Class(#vf_eltxtl) Name(#uGridName)
Define_Map For(*input) Class(#vf_elnum) Name(#uColNumber)
Define_Map For(*output) Class(#vf_eltxtl) Name(#uColName)
#ucolnumber -= 1
#myscreen_wrapper.getvalue From(#uGridName.value + ".Columns(" +
#uColNumber.asstring + ").Name") Value(#ucolname.value)
Endroutine
Mthroutine Name(uGetCellValue)
Define_Map For(*input) Class(#vf_eltxtl) Name(#uGridName)
Define_Map For(*input) Class(#vf_elnum) Name(#uColNumber)
Define_Map For(*input) Class(#vf_elnum) Name(#uRowNumber)
Define Map For(*output) Class(#vf eltxtl) Name(#uCellvalue)
#ucolnumber -= 1
#myscreen wrapper.getvalue From(#uGridName.value + ".Columns(" +
#uColNumber.asstring + ").Cells(" + #uRowNumber.asstring + ").Text")
Value(#ucellvalue.value)
Endroutine
Mthroutine Name(uTerminate) Options(*REDEFINE)
* Clean up the colelction of fields on the panel
* Do any termination defined in the ancestor
Invoke Method(#Com_Ancestor.uTerminate)
Endroutine
Evtroutine Handling(#PHBN 1.Click)
Invoke Method(#myscreen wrapper.MakeRampAvailable)
Foraction(ShowDiskStatus)
```

Endroutine End_Com

Programming Techniques

This section shows programming techniques to help you overcome common application design issues and to easily integrate advanced functionality in your RAMP applications.

Defining Screens

- Handling a Single Screen which Shows Multiple Modes
- Handling Multi-5250 Screen Data Entry

Programming

- Short-circuiting Navigation
- A Command Handler Tab with Many 5250 Destinations
- Advanced Prompting
- A RAMP Design Approach Using a Single Junction Point (SJP)
- Using HIDE_CURRENT_FORM to manage access to command handler tabs

Handling a Single Screen which Shows Multiple Modes

In System i applications it is possible that a single screen handles multiple modes.

For example, an application can have a single screen which allows ADD, CHANGE, DISPLAY and DELETE.

To be able to handle such screens in RAMP, the screens must have a unique screen ID in all the modes they can appear in and they must be uniquely defined as separate destination screens using newlook Designer.

If the mode is displayed on the screen, you can include it in the screen ID in newlook Identify so that the screen ID automatically changes according to the mode.

Also, the attribute bytes (the grey squares) surrounding a field in newlook Identify distinguish whether a field is input capable or not. Newlook is able to recognise the state of the attribute bytes and they can be used as part of the screen ID.

For example if you have two identical screens where the only difference is that one screen is input capable and the other one only displays information, you can use a single field with the surrounding attribute bytes as the screen ID.

Handling Multi-5250 Screen Data Entry

In this scenario three 5250 screens are used to input new orders like this:

Screen A	Order Number : Customer Number :
Screen B	Product Quantity Price
Screen C	State Tax : Federal Tax :

We need to do is to make them all work on a single command handler tab like this, handling the "New" command for business object "Order":

Screen A	30			
Order Number : Customer Number :	Next Sc	reen B		
C#17	Product Quantity Pr	Previous Next		en C
			State Tax : Federal Tax :	Previou Save

Start another new Order

Here's an outline of the steps required to do this and some ideas about how this common type of 5250 interaction might be modernized:

- First, enroll Screen A, Screen B and Screen C in RAMP as destinations.
- Now, link Screen A to the "New" command in the "Orders" business object.
- Do not link destination screens B or C to anything. They are defined as destinations only so that RAMP can control their function keys and execute scripts associated with them. They are never directly accessed by any command, so their INVOKE scripts will never be used.
- Now examine and think about the INVOKE, BUTTON and RETURN scripts associated with each of screen A, B and C as follows:

	Screen A	Screen B	Screen C
INVOKE script	Should be okay as generated.	Should never be used, so delete any script lines or add an error message.	Should never be used, so delete any so lines or add an error message.

BUTTON script	Successful Enter/OK button usage should cause Screen B to appear.	Successful Enter/OK button usage should cause Screen C to appear	After successful Enter/OK button you would probably script for, or, try usin NAVIGATE_TO_DESTINATION("S A") to start another new order being in
RETURN script	Should be okay as generated	Make sure it navigates back to the correct junction and not back to screen A.	Make sure it navigates back to the cor junction and not back to screen B.

The preceding table mentions "Successful Enter/OK button usage". This means that your script does a SENDKEY(KeyEnter) and then possibly checks that everything went as expected.

As generated, default scripts probably would handle the Screen A -> B -> C flow automatically, but it might be useful to understand how this happens so that you can modify the behavior. Consider this modified partial BUTTON script:

/* The user has clicked the OK button or pressed the enter key on **screen A** */

case KeyEnter:

/* Send the enter key on Screen A to the System i 5250 server */

SENDKEY(KeyEnter);

/* Now handle the screen that results (ie: after sending the Enter key) */

```
switch ( CURRENT_FORM() )
```

{

/* If **screen B** is now being displayed we have *advanced* to the 2nd screen. */

/* There is nothing more to do as screen B's scripts will now take over. */

case "Screen B": break;

/* If **Screen A** is still being displayed the user has probably made a data entry error */

case "Screen A":

ALERT_MESSAGE("Please correct the data entry errors and click OK again.");

break;

/* If we reach here, then some unexpected screen is being displayed */

default:

```
HIDE_MESSAGE("Unexpected screen " + CURRENT_FORM() + " encountered." );
```

break;

```
}
```

break;

• Finally, think about adding, re-labeling or changing buttons and function keys so as to get a more Windows like "Previous" -> "Next" -> "Save" flow going on between screens A, B and C. Possibly something like this:

	Has a "Next" Button	Has a "Previous" Button
Screen A	Yes - probably re- labels existing Enter/OK button. Script sends KeyEnter to advance	No.

	to screen B	
Screen B	Yes - probably re- labels existing Enter/OK button. Script sends KeyEnter to advance to screen C	Yes - probably re- labels something like F12 to cause Screen A to be redisplayed (the 5250 application would need to support this of course)
Screen C	No - Existing Enter/OK button is probably re-labeled as ""Save" instead. Script probably sends KeyEnter to advance to screen A to start a brand new order (after saving the current one).	Yes - probably re- labels something like F12 to cause Screen B to be redisplayed (the 5250 application would need to support this of course)

If a pop-up message is displayed when leaving Screen A B or C

Sometimes a 5250 pop-up message is displayed when leaving Screen A, B or C (either by pressing F12 on System i or by selecting another object in the Framework) asking to confirm the changes, and the response to the message takes the user to different screens. For example if the answer is Yes the user might be taken to the nearest junction, but if the answer is No the user might be taken back to the entry screen (A B or C).

The easiest solution to this is to ignore the popup by defining it in newlook as a full display (see How to Turn Pop-Ups into Full Screens) and defining it to RAMP as a special screen. The script for the special screen can set the value of the Response field to Yes and send the Enter key. In this way, the assumption is that the user is cancelling the entry when they select something else in the Framework.

In addition a Cancel button on each of the entry screens can be set to send an F12 key to get to the special screen and back to the junction, and the pop-up can be hidden. Alternatively the Cancel button script for the F12 key could contain a

javascript confirm() function, which will display a confirmation box to the user:

```
var answer = confirm("Confirm cancellation of changes?")
if (answer == true)
{
    /* user pressed the ok button on the confirmation box to cancel, so send
the F12 key */
    SENDKEY(KeyF12);
    /* if popup is defined as a special, let the special eliminate the screen */
    HIDE_CURRENT_FORM("Entry successfully cancelled.");
    }
    else
    {
        /* user pressed the cancel button on the confirmation box, so don't do
anything */
    }
```

Short-circuiting Navigation

Sometimes 5250 enquiry style transactions use a 5250 screen that loop on <enter key values> -> <display data> -> <enter key values> -> <display data> interactions.

Typically these screens have an INVOKE script that is structured along these lines:

var strAccessMenu = "SomeMenuScreenName"; var strThisEnquiry = "SomeCherryEnquiryScreenName";

/* Navigate to the menu that will provide access to the enquiry function */

NAVIGATE_TO_JUNCTION(strAccessMenu); if (!(CHECK_CURRENT_FORM(strAccessMenu, "Could not display

menu", strAccessMenu)) return;

/* Invoke the screen from the access menu */

SETVALUE("uMenuOptionField","7"); /* Say */
SENDKEY(KeyEnter);

if (!(CHECK_CURRENT_FORM(strThisEnquiry,"Could not display enquiry screen", strThisEnquiry)) return;

/* Enter the appropriate key value and display the enquiry details */

SETVALUE("uKeyValueField",objInstanceList.Akey1[0]); /* Say */ SENDKEY(KeyEnter);

if (!(CHECK_CURRENT_FORM(strThisEnquiry,"Could not display", strThisEnquiry)) return;

/* Finished */

When this transaction is used repeatedly (for example by clicking down through an instance list or orders, products, policies, etc) you can sometimes shortcircuit the navigation logic by a simple script change along the following lines. This change produces less 5250 screen interactions giving a faster response for your end-users:

```
var strAccessMenu = "SomeMenuScreenName";
var strThisEnquiry = "SomeCherryEnquiryScreenName";
```

/* We only need to navigate if we are not already at the screen */

```
if ( CURRENT_FORM() != strThisEnquiry )
{
```

/* Navigate to the menu that will provide access to the enquiry function */

NAVIGATE_TO_JUNCTION(strAccessMenu); if (!(CHECK_CURRENT_FORM(strAccessMenu, "Could not display menu", strAccessMenu)) return;

/* Invoke the screen from the access menu */

```
SETVALUE("uMenuOptionField","7"); /* Say */
SENDKEY(KeyEnter);
if (!(CHECK_CURRENT_FORM(strThisEnquiry,"Could not
```

display enquiry screen",strThisEnquiry)) return;

}

/* Enter the appropriate key value and display the enquiry details */

SETVALUE("uKeyValueField",objInstanceList.Akey1[0]); /* Say */ SENDKEY(KeyEnter);

if (!(CHECK_CURRENT_FORM(strThisEnquiry,"Could not display", strThisEnquiry)) return;

/* Finished */

A Command Handler Tab with Many 5250 Destinations

You can associate many 5250 destination screens with a single command handler tab. There are many uses for this capability and it may be used to overcome some common application design issues.

For example, imagine that you have five different 5250 destination forms that each request report production criteria and then submit the report to batch. Let's call these five different 5250 screens uReport1, uReport2 uReport5.

In prototyping this application you might approach handling these five different reports in a number of ways:

Too Many Business Objects

Each report is defined as a unique business object named "Report 1" through "Report 5". In this case the application navigation tree might be structured like this ...



When the user clicks on one of the reports the entire right hand side of the form would display the reports associated 5250 form.

Too Many Command Tabs

You define a single business object called "Reports" which has five associated commands or actions called Report 1 Report 5. In this case the application navigation and command handler tabs might be structured like this ...



When the user clicks on a report tab the associated 5250 form would appear on the tab. One of the tabs would probably be a default.

There are a number of issues with these approaches:

- The first approach consumes too many business objects
- The second approach consumes too many commands (or actions)

What do you do if there are 50 or 500 different types of reports?

Solution: Dynamic Command Tab

The answer may be to use a single business object named, for example *Reports* with a single dynamic command handler tab named *Submit Report Request*. For example, here is the Reports business object set up to show two tabs. The first is "Submit Report Request" and the second is "View Spool Files" which might be used to display the output of report batch jobs in a variety of different ways.



In this example we are only interested in the "Submit Report Request" command handler tab because we need, at execution time, to dynamically vary which 5250 destination screen actually appears on it.

So how can you vary which 5250 screen appears on this single tab? There are two main ways this is done:

- A User Controlled Command Tab with Many Destinations
- A Program Controlled Command Tab with Many Destinations

Limitations

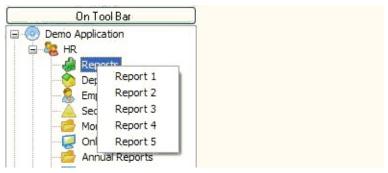
- Using the Framework SWITCH facility to switch to a command handler with many 5250 destinations is not supported.
- Any command handler using this option must be in the main Framework window, not in a separate pop-up window.

A User Controlled Command Tab with Many Destinations

You can associate several destination screens with a command handler, in which case the Framework automatically shows a window to allow the end-user decide which screen to use:

- Create the Reports business object
- Make sure the Reports business object does not have any filters and is set up so that it uses up the entire viewing area on the right hand side of the main form.
- Give Reports a single business object level command handler named "Submit Report Request". Make it the default command.
- Define the five 5250 destination forms in the normal manner.
- Associate all five 5250 destination forms with the Submit Report Request command handler tab. As you do this the RAMP tool will notify that you are associating multiple destinations with a single command handler tab.
- Execute the application.

Whenever the Submit Report Request command tab needs to be displayed it detects that it has multiple 5250 destinations and asks the user to choose which one they would like to use:



A Program Controlled Command Tab with Many Destinations

You can create a program that controls which screen is displayed on the command tab. This is slightly harder to set up but is more easily expanded.

Create the Reports business object

Give Reports a single instance level command handler named "Submit Report Request". Make this the default command.

In the business object Reports create an invisible filter that fills the instance list with the five report names. Make sure to include AKeyN and/or NKeyN values that identify the associated report. For example:

```
BEGIN_COM ROLE(*EXTENDS #VF_AC007) HEIGHT(182) WIDTH(326)
Mthroutine uInitialize Options(*Redefine)
```

#Com_Owner.avHiddenFilter := TRUE

#avListManager.ClearList

Invoke #avListManager.AddtoList Visualid1('Report 1') Visualid2('Daily production report') AKey1('uReport1') NKey1(1)

Invoke #avListManager.AddtoList Visualid1('Report 2') Visualid2('Monthly production report') AKey1('uReport2') NKey1(2)

Invoke #avListManager.AddtoList Visualid1('Report 3') Visualid2('Overloaded production report') AKey1('uReport3') NKey1(3)

Invoke #avListManager.AddtoList Visualid1('Report 4') Visualid2('Monday Morning Management Report') AKey1('uReport4') NKey1(4)

Invoke #avListManager.AddtoList Visualid1('Report 5') Visualid2('Daily production report') AKey1('uReport5') NKey1(5)

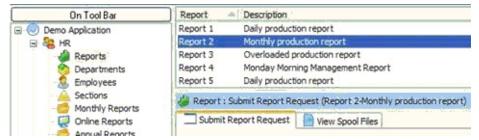
* Instance list updating has been completed

INVOKE METHOD(#avListManager.EndListUpdate)

Endroutine

End_Com

The instance list and command handler tabs are presented to the user like this:



When the user clicks on a report in the instance list the associated 5250 destination screen is displayed on the tab

Define the five 5250 destination forms in the normal manner.

Associate just the first 5250 destination forms (eg: uReport1) with the "Submit Report Request" command handler tab.

Say the numeric instance list key value NKey1 contained the requested report number then you could change the uReport1 INVOKE_SCRIPT to be like this:

```
/* See is the report number in the instance list is for some other report */
/* If it is then "reroute" this request to correct 5250 destination form */
switch ( objListManager.NKey1[0] )
```

```
{
```

```
case 2: NAVIGATE_TO_DESTINATION("uReport2"); return;
case 3: NAVIGATE_TO_DESTINATION("uReport3"); return;
case 4: NAVIGATE_TO_DESTINATION("uReport4"); return;
case 5: NAVIGATE_TO_DESTINATION("uReport5"); return;
}
```

```
/* Normal navigation logic to handle report number 1 */
```

```
NAVIGATE_TO_JUNCTION("whatever");
```

```
Etc, etc .....
```

If the alphanumeric instance list key value AKey1 contained the requested 5250 destination screen's name then you could change the uReport1 INVOKE_SCRIPT like this:

```
/* See is the 5250 screen name is this screen's name */
/* If it is then "reroute" this request to correct 5250 destination form */
if (objListManager.AKey1[0] != "uReport1")
{
    NAVIGATE_TO_DESTINATION(objListManager.AKey1[0]);
    return;
}
```

```
/* Normal navigation logic to handle this screen */
NAVIGATE_TO_JUNCTION("whatever");
Etc. etc
```

```
Etc, etc .....
```

Using this Approach in other Situations

This is example shows how to dynamically choose to present five different 5250 reporting screens onto a single command handler tab.

The choice may be made by the user or logic you write into a script.

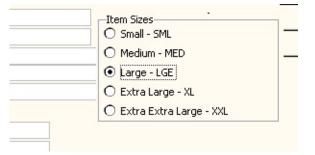
You should now understand:

- That if there were three different types of "Orders" in an "ERP" application (International, National and Local, say) that you cold use this approach to cause three different 5250 destination screens to be displayed on a single command handler tab named "Details".
- That the instance list can be used to dynamically create a "menu" of 5250 destination forms.
- That not all 5250 destination screens need to be formally attached to a command handler tab. They can be dynamically attached (ie: displayed) on tabs by logic imbedded in a navigation script by using the NAVIGATE_TO_DESTINATION() function.

Advanced Prompting

You can easily provide advanced prompting in your 5250 RAMP screens by associating simple Visual LANSA forms with fields.

For example you could create a Visual LANSA form to show different item sizes as a set of radio buttons and then associate this form with an Item Size field in the RAMP screen to return its value:



The prompter forms give you access to all the advanced Visual LANSA features such as radio buttons, sortable tree and list views, etc.

Unlike System i prompting, Visual LANSA prompter forms do not necessarily cause any interaction with the System i server which makes them fast.

Moreover, advanced prompting can be used to provide functionality that is not possible on a 5250 device. For example, a phone number prompter could display a phone number search web site and when the user chooses a phone number, place it's value back into the 5250 screen.

Other Uses for Prompter Forms

Prompter forms can also be used in various ways for sophisticated Windows desktop integration. For example they might:

- Prepare and send an overdue payment e-mail.
- Submit a credit reference check via an internet site or a web service.
- Extract information from the System i server, create a MS-Excel spread sheet, then start MS-Excel to display the spreadsheet information.
- Display a linked or associated web page.
- Display a linked or associated PDF document.
- Do any other form of advanced Windows desktop integration that you can dream up.

The advanced prompter forms are designed as an easy way integrate sophisticated functionality to subsets of information on the 5250 screen. Of

course entire new RAMP screens can be added to a RAMP application any time to handle all desktop integration requirements.

Using Prompter Forms

Creating Prompter Forms

Create prompter forms as normal VL forms.

Their Ancestor property must be se to VF_AC017 so as to inherit standard behavior.

Associating Prompter Forms with Fields

To associate prompter forms with fields, open the RAMP window and click on the session object in the navigation tree. The Session properties are displayed:

	ult NewLook Layout [pht 380 W	Dimensions /idth 700	Top Left	
ор	Mask Height	Botton	n Mask Height	
	ck framework when u cial Field Handing			
beo		Function Key	VI. Handler (class VF. AC017 object)	~
	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)	
1	5250 Field Name txtSTATE	F4	P_STATE	
1 2 3	5250 Field Name	F4 F5	in a second s	
1	5250 Field Name txtSTATE	F4	P_STATE P_PHONE	

The Special Field Handling area is used to define the forms to be associated with fields.

The two entries in the example indicate that:

If an input field named txtSTATE is on any 5250 destination form, and it is where the cursor/focus is, and the user presses function key F4 (or the equivalent button) then the VL form named P_STATE is to be invoked to handle the request.

If an input field named txtPHONE is on any 5250 destination form, and it is where the cursor/focus is, and the user presses function key F5 (or the equivalent button) then the VL form named P_PHONE is to be invoked to handle the request.

How do Advanced Prompter Forms Work?

Whenever the user performs the actions required to invoke one of the VL forms

the following happens:

- The HANDLE_PROMPT Function in the script is invoked to show the prompter form associated with the field. Optionally additional information can be passed to the form using this function.
- If the form has not been used already in the session it's uInitialize method is invoked. This allows it to do first time processing.
- The values of all the named fields on the current 5250 destination form are extracted and made available to the VL form.
- The VL form's uShow method is then invoked so that it can prepare and position anything that it wants to show to the user.
- When the user makes a selection, the VL form can alter the value of any named field on the current 5250 destination form.

Are any Examples Provided to Learn More about this Topic?

Yes, you should be able to find the following Visual LANSA forms in your repository:

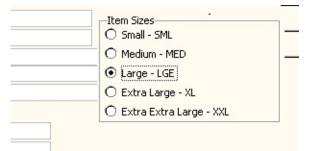
Combo Box

DF_PRM01 prompts using a combo box of US states like this:

ALABAMA - AL ALASKA - AK AMERICAN SAMOA - AS	
ARIZONA - AZ ARKANSAS - AR	
CALIFORNIA - CA	

Radio Buttons

DF_PRM02 prompts using a set of product size radio buttons:



List with Columns

DF_PRM03 generically prompts for employees by name:

LGE		Pror	npt	
GARY	DE 052503 0	romoting Evample	2	
14 Hutchins Crescent,	🚽 🗖 DF_P52503 - Prompting Example 3 👘 📃 🔟 🔀			
PADSTOW HEIGHTS.	Employee Surname	В	Search	
NSW.				
2016	Employee Number	Employee Surname	Employee Given	Post / Zip Code
456 1524	A1031	BLAKE	JOHN	0
151 4583	A0090	BLOGGS	FRED JOHN	0
83/03/01	A3564	BROWN	FREDDY	0
	A0070	BROWN	VERONICA	0
0/00/00				
INF 🗔				
02				
25,600.04				
2.133.33	P		1	1 - 11

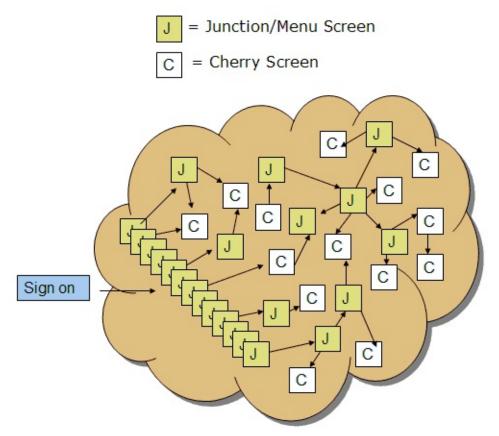
Tree

DF_PRM04 prompts department and section information using a tree:

Post / Zip Code	2016	
Home Phone Number	456 1524	
Business Phone Number	151 4583	
Start date (YYMMDD)	83/03/01	
Termination Date (YYMMDD)	0/00/00	
Department Code	INF B ADMINISTRATOR DEPT	
Section Code	02 B INTERNAL AUDITING	
Employee Salary	B FLEET ADMINISTRATION	
Monthly Salary	B-GROUP ACCOUNTS	
Start Date (DDMMYY)	1/03/83	
Termination Date (DDMMYY)	0/00/00 PURCHASING	
	ACCOUNTING	
	DEVELOPMENT	
	B-LEGAL DEPARTMENT	
		~

A RAMP Design Approach Using a Single Junction Point (SJP)

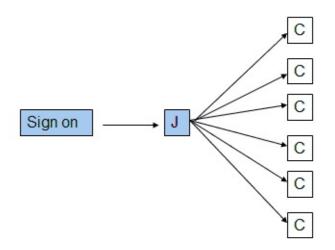
A complex 5250 application that RAMP is being applied to may be visualized like this:



A 5250 user signs on and navigates around a cloud of menus/junctions to reach the "cherries" (5250 destination screens) where they do useful work.

The RAMP choreographer is able to follow these navigations and working with it you can define the various navigations required to move around in the cloud. To a RAMP developer the identification of the junctions and the generation of their navigation scripts may be a time consuming and rather mundane job.

From the RAMP developers point of view the whole process would be easier to handle if the 5250 application was actually structured like this:



Here a single junction point (or program) controls access to every 5250 destination screen.

If the 5250 application was structured this way then designing a RAMP application would be simpler and faster because:

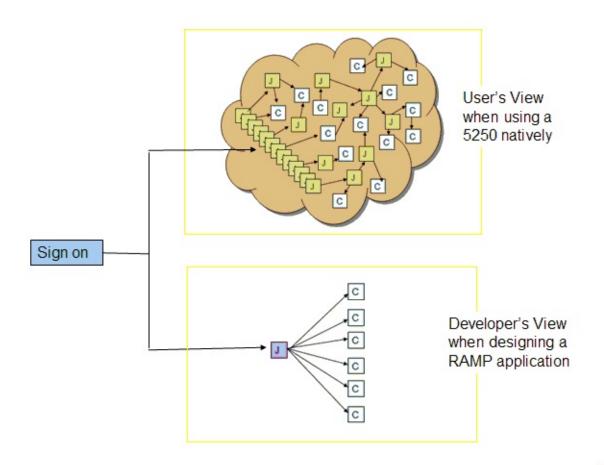
- Only a single junction needs to be defined and scripted.
- The invocation scripts for the destination screens are simpler and standardized.

This rest of this section describes ways that you might set up this type of view of a 5250 application.

This approach is called the *Single Junction Point (SJP)* model.

The SJP model cannot be applied to every type of application, but where it can be applied it may represent a saving in the time taken to develop a RAMP application.

Essentially a SJP approach means that two different views of an application exist:



To make this programmatic view of the world the System i 5250 program needs to already exist or to be created.

Let's call this special program the SJP (Single Junction Point) program

A kind of specific already exists on all System i system.

It is a program called QCMD (or Command Entry Display) and from it almost any 5250 application can be invoked in some direct or indirect way. However using QCMD is not acceptable to many sites for security reasons, so the rest of this material discusses various ways you might create your own specialized

SJP K and some of issues and additional benefits that might arise.

How does an SJP work?

Is an SJP really that simple in a real application?

Can SJP do the other useful things?

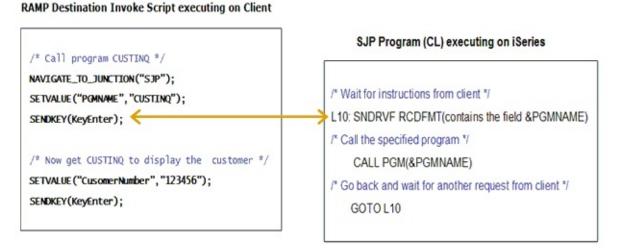
Does SJP have to be CL (Control Language) program?

What other issues might impact the use on an SJP approach?

How does an SJP work?

An SJP program provides generic access to the destinations that are available to a RAMP application. An SJP is not designed to talk to a user, it is designed to talk to a RAMP script.

A simple SJP and RAMP script might work together like this:



The RAMP script example used here is associated with a command handler that wants to display the details of a customer using a 5250 program named CUSTINQ.

When it starts to execute it first navigates to the junction screen named JSP.

This causes the SJP program to displays its 5250 screen.

It then sets the field PGMNAME to value "CUSTINQ" and sends the enter key.

This causes the SJP program to receive the screen back.

The CL field & PGMNAME in the SJP program now contains the name "CUSTINQ".

Program CUSTINQ is then called using a generic call.

The RAMP script then gets the program CUSTINQ to display customer number 123456.

Using this simple SJP hundreds of destination screen scripts could be created to access all sorts of System i 5250 programs, providing that they all have a simple CALL interface.

Is an SJP really that simple in a real application?

Probably not. Often the programs being called required simple (and sometimes complex) parameters to be passed to them and amongst them.

However, in this style of application design, groups of programs usually fall into large application groups that share a *common parameter protocol*.

By adding an REQUEST_TYPE (say) field to the information exchanged between RAMP scripts you can easily accommodate different program parameter protocols along these lines (logic is in pseudo code):

WRITE and READ the 5250 screen containing PGMNAME and REQUEST_TYPE

DOWHILE (REQUEST_TYPE not equal to "SIGNOFF")

CASE of REQUEST_TYPE WHEN = "CALLP1" CALL PGM_NAME using calling protocol 1 for parameters WHEN = "CALLP2" CALL PGM_NAME using calling protocol 2 for parameters WHEN = "CALLP3" CALL PGM_NAME using calling protocol 3 for parameters <etc> ENDCASE

WRITE and READ the 5250 screen containing PGMNAME and REQUEST_TYPE ENDWHILE

If you are used to RPG and CL programs you might not be aware just how flexible the IBM i program call interface is. Program parameters are just areas of memory and passed between programs as pointers. You might not know:

- Parameters do not have to be the exact length the called program defined. They just need to be as long or longer, which makes sharing and reusing a small set of parameter variables in a SJP quite simple.
- You can pass a program more parameters than it actually requires. The extra ones are generally ignored, which means you can have very few actual

CALL commands in your program.

• You could directly pass parameter values from you RAMP scripts to the SJP and pass them into the called programs. You could also get retuned parameter values back into the script again using this approach. This means your RAMP scripts can call batch style programs as well.

Can SJP do the other useful things?

It could be designed to do almost anything. For example it can provide a very flexible and generic interface to IBM i command like this:

WRITE and READ the 5250 screen containing PGMNAME, REQUEST_TYPE and COMMAND

DOWHILE (REQUEST_TYPE not equal "SIGNOFF")

CASE of REQUEST_TYPE WHEN = "CMD" CALL QCMDEXEC (COMMAND 256) WHEN = "CALLP3" CALL PGM_NAME using calling protocol 3 for parameters <etc>

Would allow your RAMP scripts to execute a CL command like this: NAVIGATE_TO_JUNCTION("SJP"); SETVALUE("REQUEST_TYPE","CMD");

SETVALUE("COMMAND","WRKSBMJOB *JOB") SENDKEY(KeyEnter);

Or

NAVIGATE_TO_JUNCTION("SJP"); SETVALUE("REQUEST_TYPE","CMD"); SETVALUE("COMMAND","SBMJOB(BATCH) CMD("CALL PRINTORDER")") SENDKEY(KeyEnter);

The 5250 screen used to communicate between a RAMP script and a SJP is really more of program data structure that a real 5250 screen that a user would ever see.

Does SJP have to be CL (Control Language) program?

No, it could be written in any program language that supports the reading and writing of 5250 screens such RPG, COBOL, C or RDML (which is really RPG anyway).

If you have LANSA programs RDML is a good choice because it makes it very easy to call LANSA processes and functions and allows access to common inter-program communications mechanism such as the exchange list and data structures.

What other issues might impact the use on an SJP approach?

One of the main ones relates to user profile and site security requirements.

You would probably not want the SJP program accessible to USERA (say) when he or she is using a normal 5250 screen.

Additionally most sites insist that USERA executes his/her IBM i job under the profile USERA so that audit, log and security information shows the "real" user (although this is disappearing as more and more "threaded" processes serving many concurrent users, such as HTTP web servers, are used on the System i server).

So how can a single user profile USERA support these different views of the world?

- When they sign on to a real 5250 session they get their normal sign-on menu.
- When they sign on via a RAMP script they get the SJP program as their main "menu"?

There are several solutions to this problem:

- Use the Program/Procedure option on the IBM i sign-on screen to specify the SJP program when logging in via a RAMP script. You would probably add some security logic to the JSP to prevent users doing this through a real 5250 interface (see point 2).
- If you use a common menu program you could alter it to detect that it is being called from a RAMP script and then call the SJP program. Equally you could display the common menu initially and use a special "hidden" menu option to call the JSP program. The JSP program could confirm that it is being accessed by a RAMP script by conducting, for example, an encrypted exchange with the RAMP script that is impossible for a real human user to perform.
- RAMP scripts could sign on initially as a generic "USERX" whose initial program is the SJP program. The SJP program then presents a screen asking for the real user profile and password, which the RAMP logon script fills in and sends back. An IBM API is then called to change the current job's user profile from generic USERX to the real user. Again an encrypted exchange that is impossible for a real user could be used to confirm access is from a RAMP script.

Using HIDE_CURRENT_FORM to manage access to command handler tabs

In this scenario a RAMP application has been created over an order processing system.

Imagine that some of the command handler tabs (and their underlying 5250 destination scripts) need to prevent users from performing actions on cancelled or completed orders.

Step 1 - Put some sort of "Code" or "Status" column into every instance list entry

Here field #ORDSTATUS is mapped into instance list column Acolumn9(). Imagine it contains values "CAN" (cancelled), "OPN" (Open), "WIP" (Being worked on) or "COM" (completed)

Invoke Method(#avListManager.AddtoList) Visualid1(#OrdNo) Visualid2(#CustlName) Akey1(#OrderNumber) AColumn9(#ORDSTATUS)

Note: AColumn9() may or may not be shown to the user as desired.

Step 2 - Put checking code into the appropriate INVOKE scripts

Here the INVOKE script for a 5250 screen that allows an order to be modified has had a check added to stop people from trying to display cancelled or completed orders

/* Get the order status from additional column 9 in the current order instance list entry */

var ORDSTATUS = objListManager.AColumn9[0];

/* If the order is cancelled or closed, prevent the 5250 screen from being displayed, and show a message as to why */

```
if ((ORDSTATUS == "CAN") || (ORDSTATUS == "COM"))
```

{

HIDE_CURRENT_FORM("Sorry, but you are not allowed to display this

order because it is cancelled or completed.");
 return;
}

/* If we reach here then it's okay to proceed to the order display screen */

NAVIGATE_TO_JUNCTION("OrderMainMenu");

<etc> <etc>

The HIDE_CURRENT_FORM("message") function causes the current 5250 screen being displayed on the command tab to be hidden and the message "Sorry, but you are not allowed to display this order because it is cancelled or completed." to appear in the center of the tab instead.

The content of AColumn9 (ie: "CAN", "OPN", "WIP", "COM") could be used anywhere in INVOKE or BUTTON scripts to limit or control user activities.

Multilingual RAMP Applications

Strings

Refer to the ADD_STRING Function and the STRING Function.

The captions show on RAMP buttons can be changed to be multilingual using the OVERRIDE_KEY_CAPTION_SCREEN Function and OVERRIDE_KEY_CAPTION_ALL Function.

Troubleshooting

Error Messages

An unexpected database error has occurred TCP/IP timeout has occurred TCP/IP host was not found The connection to <newlook server name> has not been defined Script cannot be generated at this time xxxxxx is an orphan script and should be deleted Error running RAMP in end-user mode (UF_EXEC) but not in design mode (UF_DESGN)

Problems

When recording:

newlook cannot be started in the RAMP Window

Keystroke is ignored

RAMP does not recognise the name of forms that I have defined recently or any other newlook definition changes

RAMP Choreographer does not recognize a screen that has a name in newlook Designer

When executing RAMP applications:

Navigation is Incorrect, but there is no error message

Strange behavior in scripts

Screen does not react when selection is changed in instance list

A Screen is not recognized

When scripting:

Subfile accessor only reads the first page

An unexpected database error has occurred

An newlook database error is displayed:



After the error you cannot run the Framework.

What does this error mean?

Some part of your system is corrupted.

Solution

First run the newlook SID file cleanup program CHKSID.exe with the -r parameter.

Then Merge Shipped Macros into newlook again.

xxxxxxx is an orphan script and should be deleted

A RAMP warning message is displayed saying that a script is an orphan script and should be deleted.

What does the message mean?

- It means the script is not used by any destination, junction or special screen.
- Since the script is not used, it should be deleted.
- This message does not impact the operation of RAMP, *it's just a warning*.
- If you get a lot of these warnings, it is likely to be a misunderstanding about use of the merge tool in a multi-developer environment.

How do you delete a script if you get this message?

- Start the RAMP tools
- Expand the script tree node and locate the script.
- Select the script and press the Delete button.
- Watch out for duplicated script names (this happens in multi-developer environment). Make sure you have the right script.

How can you get an orphan script?

• The most likely way is by using the merge tool to merge in a brand new script all by itself without merging in the parent destination, junction or special screen as well.

When would you use the merge tool to just merge in a single script without also merging in its parent destination, junction or special screens as well?

• Normally you would only do this when you have previously merged in the parent object and its associated scripts and are just wanting to merge in a single updated script. You should never do this on an initial merge or you risk creating orphan script(s).

How should you approach merging RAMP screens and their associated scripts produced by multiple developers?

Assuming that the high level Framework design objects, that is applications, business objects, commands and command handlers (tabs) have been set up by the master designer and all developers are working from the same model (that is the developers just define the RAMP screens and scripts and then link them up to the pre-defined command handler tabs):

- The sender should add the screens that they have produced to a merge list. This should automatically include the associated scripts.
- The sender should also add to the same merge list all the command handler(s) that have been modified by being linked up with RAMP destination screens.
- The receiver should merge everything into the master Framework. The command handlers should be handled as updates/replacements and the RAMP objects should be new objects. In no case should new GUIDs be assigned.

How can you get scripts with the same name?

In a multi-developer environment if two developers create scripts, you may end up with two scripts named for example INVOKE_SCRIPT_16.

If the work of these developers is merged together, this situation may be confusing to the developers, but it is not confusing to RAMP because to RAMP the script name is just a caption. Internally RAMP recognizes and executes scripts by their unique GUID.

Developers can change the default script names to avoid this confusion.

Navigation is Incorrect, but there is no error message

Navigation produces no error but it doesn't do what it should.

When does this problem happen?

This can happen when a field in a RAMP screen which is required for navigation has not been given a name using newlook Designer and a script is generated using RAMP's tracking facility.

This is because scripts produced by RAMP tracking ignore actions taken in unnamed fields.

Solution

Use newlook Designer to give fields you intend to use in the navigation a name. When the field has a name, make sure to run the navigation again so that the tracking facility can detect it and use it in the script.

Keystroke is ignored

The keystroke is ignored when recording screens.

When does this problem happen?

This happens when one and the same screen is used for both display and edit. The screen has the same name in newlook.

When the key is pressed, RAMP detects that the screen arriving is the one currently showing because it has the same name and ignores it so the keystroke is lost.

Solution

The solution is to make display and edit two different screens, in other words identify them and name them as two different screens using the newlook Designer. One with the fields as output (display) and another one with the fields input capable.

RAMP does not recognise the name of forms that I have defined recently or any other newlook definition changes

When does this problem happen?

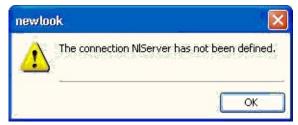
This happens when newlook is not aware of the changes.

Solution

- Check that the form object is named when in newlook designer mode.
- Check that the name of the form object (when in newlook designer mode) does not have any trailing spaces
- If this happens on a RAMP web development PC that has downloaded a newlook deployment package from a RAMP website recently, it could be that the RAMP choreographer (the newlook activeX) looking at the .sid file from the downloaded package. This will be different than the local sid file that holds the newlook screen definitions that you have made since the package was created. RAMP should not be tested on development PCs.

The connection to <newlook server name> has not been defined

Connection fails with following error:



Cause

In the Framework server definitions there is a server of type newlook with name <server name> without IP address or Port fields defined, but there is no session with the name is defined to newlook.

If you specify a newlook server in the Framework, but leave the IP address and port fields blank, the Framework assumes that there is a permanent connection with the corresponding name defined in newlook.

Solution

Option 1:

If you want to define a permanent connection in newlook:

- Start newlook 8.0.
- Click on the Session menu and select Connect. The connect dialog shows all the defined connections.
- Use the newlook connection wizard to define a new connection using the name you have used in the Framework server definitions.

For more information see Verify newlook Installation.

Option 2:

If you want to change the Server Name of the server definition to an existing newlook connection in the Framework:

- Display the Servers tab in the Framework
- Locate the server with name <server name> and change the name.

Server Type:	Newlook	X
Server Name:	NIServer	

Option 3:

If you do not want to define a permanent connection in newlook, fill in the IP address and Port Number in the Framework server details for the newlook server:

- Display the Servers tab in the Framework
- Locate the server with name <server name>
- Fill in the IP Address and Port Number fields.

For more information refer to Configure RAMP.

Script cannot be generated at this time

Script generation fails with an error saying the script cannot be generated at this time:

Solution

Check when the screen was defined in Newlook that:

- The Name doesn't exceed 265 characters.
- There are no trailing spaces in the name.

Is recommended that only characters A - Z are used and blanks or spaces are not used anywhere in the name.

Also note that screen names are case sensitive.

TCP/IP timeout has occurred

Connection fails with following error:

Run-Tim	e Error	
07	The following run-time error has occurred:	
	Macro: VF_Macro Action: Connect (Line 3)	
	Error R1006:	
	10060 - A TCP/IP timeout has occured.	<u>A</u>
		1.11
		<u>(X)</u>
	Choose one of the following:	
	C Continue running the macro	
	Abort the macro	
	arepsilon Edit the macro to correct the error.	
۲		ОК
		-

Cause

For the Server Type newlook in the Framework server definition, the IP address or the port number or both are incorrect.

Solution

In the Framework Server Details tab, locate the definition of the newlook server you are trying to connect to. Make sure the IP address and the port number are correct.

Note that if you specify a Server which has a connection defined in newlook, you can leave the IP address and port number fields blank in the Framework Server Details tab.

For more information refer to Configure RAMP.

TCP/IP host was not found

Connection fails with following error:

Run-Tim	e Error	
07	The following run-time error has occurred:	
	Macro: VF_Macro Action: Connect (Line 3)	
	Error R1006:	
	11001 - The TCP/IP host was not found.	<u>~</u>
		<u></u>
	Choose one of the following:	
	C Continue running the macro	
	Abort the macro	
	\mathbb{C} Edit the macro to correct the error	
۲		OK

Cause

An invalid host name has been specified in the IP address for the newlook server in the Framework Server Details tab.

The IP address can be specified in the form nnn.nnn.nnn or as a host name. This error points to an incorrect host name.

Solution

In the Framework Server Details tab, locate the server of type newlook that you are trying to connect to. Make sure the IP address contains a valid host name.

Note that if you specify a Server which has a connection defined in newlook, you can leave the IP address and port number fields blank in the Framework Server Details tab.

For more information refer to Configure RAMP.

Screen does not react when selection is changed in instance list

When an entry is selected in the instance list, the RAMP screen does not reflect this change and instead shows the data for the entry that was first selected.

When does this problem happen?

You recorded the invoke script of the destination screen, but you have not changed the value parameter of the SETVALUE Function from the recorded hardcoded value to a substitution value.

Another possible cause is that the value in the SETVALUE function has been enclosed in quotes in which case it is interpreted as a literal, not as a substitution value.

For example this example is wrong:

```
SETVALUE("UtxtMachine","objListManager.AKey1[0]");
```

Solution

Make the value parameter of the SETVALUE function a substitution value and make sure it is not surrounded by quotes:

SETVALUE("UtxtMachine", objListManager.AKey1[0]);

For more information:

- Watch the tutorial movie Link the Selected Employee in the Instance List with the Display Employee Screen 4 minutes
- See the topic Interacting with Instance Lists in Scripts
- See the topic Replacing Hardcoded Employee Number with Current Instance List Entry.

A Screen is not recognized

The Framework fails to recognise a screen, this may happen always or occasionally.

The Trace for the screen looks like this:

VF_CH006 Screen named has arrived and is being processed

VF_CH006 Screen named in a form not defined to the framework.

VF_CH006 Signal LockFramework is being queued.

VF_CH006 Signal LockFramework has been queued.

VF_CH006 Screen named is not a destination do function keys will be left unchanged.

Solution

Check that:

- The form object is named in newlook design mode.
- The screen ID area covers only static elements in newlook Identify mode. In other words, the ID area must not cover field data, browselist data, system date/time, System i machine name, User profile or other data that can change.

newlook cannot be started in the RAMP Window

newlook client (designer) works in its own window, but not inside the RAMP window.

When does this problem happen?

The most likely cause is that you do not have a Standard Edition or liteclient newlook licence on the server.

Solution

Check Licensing Requirements to make sure you have the correct licences.

Subfile accessor only reads the first page

If you are using the RAMP subfile accessor, and you believe it should scroll through all the pages in the subfile looking for a row, but it never seems to read beyond the first page,

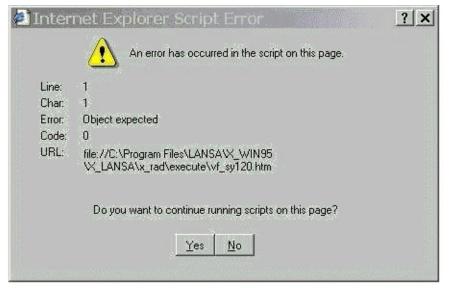
Solution

Check that:

- PageUp and PageDown function keys are working
- newlook recognises the subfile marker. Check this by going into newlook Identify, and then select the area containing the plus (+) sign that indicates there are more subfile entries, and ensure that this is ticked as SubFile Marker.

Error running RAMP in end-user mode (UF_EXEC) but not in design mode (UF_DESGN)

You can run your RAMP application in Design mode but you get an error like this when you try to run it in End-User mode:



Why does this problem happen?

The main difference between running RAMP in design mode and running it end-user mode is the way javascript is executed.

In design mode, javascript is reloaded each time the Framework is saved if there has been a change affecting RAMP. Each time the javascript is reloaded, the object properties are re-set. And each time the Framework is saved, if RAMP is enabled and has changed, a set of javascript files called <system prefix>Nodes_nnnnnnnnnnnnnnnnnnns, is are generated, one for each session

where the nnnnnnnnnnnnnn part is the session identifier.

These files are the ones used in end user mode. They represent the screens and scripts written out as javascript at the time the Framework was saved.

In end-user mode, these files are loaded once for each session. Each one of the javascript functions in the file is called only once during session start up. This method speeds up the start up time of RAMP in end-user mode considerably as opposed to design time.

When RAMP is executed without errors in design mode but with errors like the above in end-user mode, the prime suspect is a syntax error in the user-defined scripts (be it navigation scripts, invoke, etc).

Solution

To find out what line of javascript has the error, you can simply load the file into a basic .HTM file.

For example create a file called test.htm with content like this:

```
<html>
<head>
<title>Untitled Page</title>
<script language="javascript" type="text/javascript" src="<your nodes.js file
here>"></script>
</head>
<body>
Hello World
</body>
</html>
```

Specify the name of your nodes.js file in the src= attribuite of the <script> tag and put Test.htm in the same folder as the javascript.

Using Internet Explorer, check your Advanced settings tab under Tools/Internet options to verify you have the "Display notification about about every script error" checked. You can then run Test.htm and you should get a script error showing the line number where the error has occured. Tip: the error is most likely to be inside a javascript function called something like this:

function ___UF___nnnnnnnnnnnnnnnnnnnnnnnnnnnnn(objScriptInstance)

which makes it a bit hard to correlate it with the actual script name. To find out exactly what this script is, do a Find in the same file of the nnnnnnnnnnnnnnn part of the function name. You should then locate the lines of javascript that define that script as an object and that will have the user name (for example oS.uScriptUserName="NAVIGATE_SCRIPT_13";)

Edit the script using the RAMP tools, correct the error and save.

RAMP Choreographer does not recognize a screen that has a name in newlook Designer

Using the choreographer in the RAMP tools, I navigate to a screen that I had previously named using Newlook Designer. The choreographer doesn't recognise it and shows a message saying *Unknown Form*.

If I start the Newlook windows client and navigate to that screen, I can verify that the screen has indeed been given a name.

Why does this problem happen?

Most likely you have executed a RAMP application that was deployed using a cab file. In such cases, the downloaded Newlook ActiveX control is registered in your machine as the cab file is automatically downloaded. Thereafter, you would be running a different newlook with a different SID file than the one you were working with before.

When the downloaded newlook version is different than the Newlook version that was installed, you can easily verify this anomaly:

- Locate a file called **nl_load_test.htm** in the partition execute folder. Double click on it and show the Help/About.
- Start the Newlook windows client from the Newlook folder and show the Help/About.

Solution

If the versions are different:

- Uninstall Newlook (a shortcut to this step is to run nlclean.exe and answer Y to all questions).
- Reboot
- Reinstall Newlook

Remember no tot test VLF.RAMP-WEB deployments onto your development PC!

Frequently Asked Questions

How is my newlook license type determined when starting newlook?

How can I use web browser windows from RAMP scripts?

How can I get the message from the bottom of the current 5250 screen into my RAMP script?

How do I handle RA (Auto Record Advance) fields?

Why does my newlook session have a message "Press SPACEBAR or ENTER to activate and use this control"?

What is the difference between newlook Designer and newlook Emulator Session?

Why should the F12=Cancel and F3=Exit buttons and function keys be disabled on every 5250 screen?

I have defined a screen as a junction, but it should be destination. How do I change it?

Do I have to identify and script every 5250 screen in my application to modernize it?

How can I get the RAMP tool to assign a fixed session?

How do I make my scripts work in multiple partitions?

How can I change the background color of RAMP screens?

How can I ~suppress the action of Alt + F4 inside Newlook?

How is my newlook license type determined when starting newlook?

In RAMP Tools

If any of the newlook server definitions in your Framework has the Use liteclient license option checked, RAMP tools start newlook requesting that a liteclient license type is used. Otherwise newlook will use a default license type.

When executing Windows Applications

When newlook is started it is associated with a server defined in your Framework.

If the server has the Use liteclient license option checked, a liteclient license type will be used. Otherwise newlook will use a default license type.

When executing Web Browser Applications

When the HTML and JavaScript flies associated with your Framework are saved a default license type is determined.

If any of the newlook server definitions in your Framework has the Use liteclient license option checked, the default will be to use a liteclient type of newlook license. Otherwise a default newlook license will be used.

You can override this default by adding +NLLiteClient=TRUE or +NLLiteClient=FALSE to the URL you use to start your web browser application. For more information see Web Application Start Options in the Framework guide.

There may be exceptions

If a developer modifies the VF_UM703.HTM (RAMP tools) or VF_SY120.HTM (RAMP Execution) start up pages they may *unconditionally force* a newlook license type to be used in all situations.

How can I use web browser windows from RAMP scripts?

Here's a really simple web browser form that accepts three input fields as arguments, displays them, allows them to be altered, then returns the altered values back to the calling RAMP script:

```
<HTML>
<HEAD>
</\text{HEAD}>
<BODY onload="BODY_Load();" onunload="BODY_UnLoad();" >
<script>
function BODY_Load() /* Map arguments passed in to web form fields */
ł
 FieldA.value = window.dialogArguments[0];
 FieldB.value = window.dialogArguments[1];
 FieldC.value = window.dialogArguments[2];
}
function BODY_UnLoad() /* Map web form fields into return values */
 var arrayRets = new Array();
 arrayRets[0] = FieldA.value;
 arrayRets[1] = FieldB.value;
 arrayRets[2] = FieldC.value;
 window.returnValue = arrayRets;
function OK_Click() /* Handle OK button by closing the web form */
 window.close();
}
</script>
<P>Input details and click OK"<br/>
<input id="FieldA" type="text"><br/>
<input id="FieldB" type="text"><br/>
<input id="FieldC" type="text"><br/>
<input id="Button1" type="button" value=" OK " onclick="OK Click();">
\langle BODY \rangle
</HTML>
```

It looks like this when displayed:

🗿 Web Page Dialog 🛛 🔀
Input details and click OK"
SMITHS
PETERS
72 Mullane Avenue,
UN

This is the RAMP BUTTON script that is used to display the web browser form. It displays the form when the user hits F5, taking the fields SURNAME, GIVENAME and ADDRESS1 from the 5250 form and then mapping them back:

```
switch (objScriptInstance.FunctionKeyUsed)
{
 case KeyEnter:
   SENDKEY(KeyEnter);
   break:
 case KeyF5:
   {
    var arrayArgs = new Array();
    arrayArgs[0] = GETVALUE("SURNAME");
    arrayArgs[1] = GETVALUE("GIVENAME");
    arrayArgs[2] = GETVALUE("ADDRESS1");
    arrayRets =
window.showModalDialog("Example.htm",arrayArgs,"dialogHeight:155px
    SETVALUE("SURNAME",arrayRets[0]);
    SETVALUE("GIVENAME",arrayRets[1]);
    SETVALUE("ADDRESS1",arrayRets[2]);
    delete(arrayArgs);
    delete(arrayRets);
   }
   break;
 default:
   SENDKEY(objScriptInstance.FunctionKeyUsed);
   break;
}
```

This is just a simple example of some of the things you can do (please note that no warranty about any of this is expressed or implied).

How can I get the message from the bottom of the current 5250 screen into my RAMP script?

Use a script like this:

```
{
    var strMessage = GETVALUE("ActiveForm.Message"); /* Get the
message into JavaScript variable strMessage */
    if (strMessage != "") ALERT_MESSAGE(strMessage); /* If a message
was retrieved, display it in a message box */
  }
```

How do I handle RA (Auto Record Advance) fields?

Some 5250 applications may use fields with an RA input attribute (Auto Record Advance). Programs that display these fields automatically press Enter when the last digit or character is entered by the user.

The RAMP choreographer cannot automatically generate a script for this situation based on your keystrokes. Instead, it will generate lines like:

/* Set up data fields on form xxx */

SETVALUE("utxtMenuOption","");

/* Send the key required to navigate to xxx */

You will need to edit the generated script, and specify both the value and the Enter key press, like this:

/* Set up data fields on form xxx */

SETVALUE("utxtMenuOption","2");

/* Send the key required to navigate to xxx */

SENDKEY(KeyEnter);

Why does my newlook session have a message "Press SPACEBAR or ENTER to activate and use this control"?

Press SPACEBAR or ENTER to activate and use this control

This message is presented by a change to the handling of Active-X controls that Microsoft introduced with Service Pack 2. They have since rescinded this patch because of the disruption it caused to existing applications. Please refer to http://support.microsoft.com/?kbid=917425 for more details.

What is the difference between newlook Designer and newlook Emulator Session?

You use the newlook Designer to identify all the screens in the application being modernized by giving them a unique name. You start the Designer by clicking the newlook Designer button in the RAMP window.

You use the newlook emulator session(which is located in the top left corner of the RAMP window) to define your screens to the Framework and to trace navigation between the screens. To start the emulator click on the message *newlook has not been started* in the message area.

Why should the F12=Cancel and F3=Exit buttons and function keys be disabled on every 5250 screen?

Have a think about how you navigate a Windows application.

I have defined a screen as a junction, but it should be destination. How do I change it?

Delete the screen definition in the 5250 screen list on the bottom left of the RAMP window. The screen will appear as undefined in the Tracking Information area.

Do I have to identify and script every 5250 screen in my application to modernize it?

No.

Typically some areas of a 5250 application are rarely used or used by very few users.

The degree of modernization you apply to an application area should be related to the area's degree of exposure to end users and to the amount of benefit that they would gain if it were completely modernized.

Modernizing a Single Screen to Provide Access to a Subsystem

In this example a 5250 menu or work with screen named uCodeTableMaint manages access to 47 different 5250 screens that handle System Code Table Maintenance (for example classic code and parameter tables such as states, companies, currencies, interest rates, etc that are used to define and control an application).

uCodeTableMaint

uCodeTableMaint could be visualized as an "application subsystem" like this:

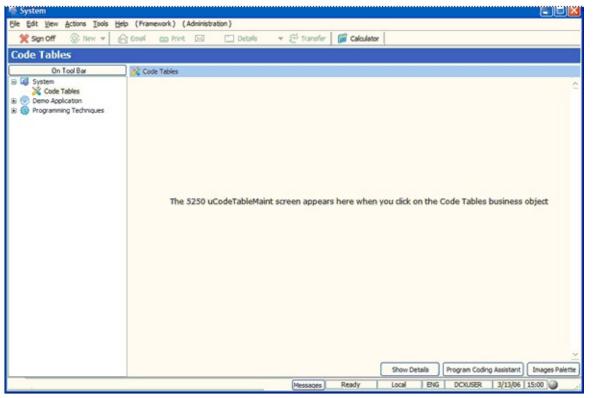
Because this application area does not need to be completely modernized, the most rapid way to modernize it is to create a single RAMP screen that provides access to the other screens.

To do this:

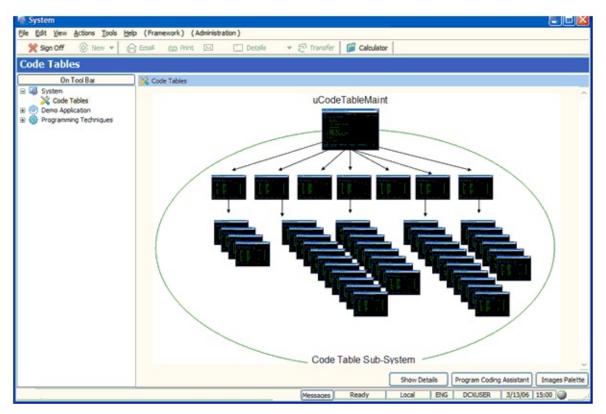
- Create a business object called Code Tables and associate with an application.
- Give it a single RAMP screen (or tab) called Maintain (say). Make sure this is an object level command and that it is the default command so that it is executed automatically every time you click on it.
- Identify and define the 5250 work with screen uCodeTableMaint to the Framework as a destination screen and associate it with the Maintain screen.

When the user clicks on Code Tables in the Framework application they are immediately navigated to the uCodeTableMaint 5250 screen.

It occupies the entire right hand side of the windows form like this:



Once the user has displayed the uCodeTableMaint screen they can then navigate around in the other 47 associated screens in the normal manner:



This is a minimal modernization of the whole uCodeTableMaint managed subsystem.

Only the 5250 screen uCodeTableMaint needed to be defined and scripted into the framework. The other 47 screens did not have to be identified nor scripted in any way.

How can I get the RAMP tool to assign a fixed session?

I want to assign a fixed session, such as Session A, for my destination screen in the Destination Screen Details. How can I do this?

You have to select the command handler so that the line it is on goes blue, not just tick the checkbox. Then you associate a session with it.

This may seem unusual, but sometimes multiple command handlers are associated with a single destination form and therefore you have to actually indicate which one you want to change the session for.

How do I make my scripts work in multiple partitions?

Replace any hard-coded references to a partition in your scripts with this piece of code:

objFramework.Partition

You can enter the code Using the Scripting Pop-up Menu: choose Current Framework and then partition.

How can I change the background color of RAMP screens?

To change the RAMP background color, edit the visual style called UF_VS006 and change the NormBackColor of the Caption and Value to Buttonface. Recompile your equivalent of UF_SYSTM, to pick up the change. The visual lansa component that displays RAMP screens will then pick up the current theme.

To change the theme within the newlook area of a RAMP screen, ensure that the shipped VF_XP_2003*.nlg and VF_XP_2007*.nlg files are present in the looksoftware directory and from within newlook, choose the matching scheme in --> Tools --> Settings --> Display --> Appearance, Settings --> Scheme and apply.

How can I suppress the action of Alt + F4 inside Newlook?

Pressing Alt + F4 when focused in the RAMP-NL command handler causes the Newlook session to terminate (the user is prompted with the "do you wish to exit" pop-up). The Framework however remains active.

You can disable the Alt+F4 key by doing the following:

- 1. Create a macro named Disable Alt+F4 Key (or name it whatever you want).
- 2. Set the Action on row 1 to be **CancelEvent**.
- 3. Then choose File|Properties|Menu from the Macro Editor.
- 4. Then assign the **Alt+F4** key as the shortcut key for this macro and your problem will be solved.

Edit Debug Run			V 14 G	× @ /	A AA	A C
					649	i 8 840 I
		- 100 €	660' Con	text:		
Label 1	Condition	ancelEvent				
z ave As					×	
General Summary	Menu					
Use the newlood	se options to spec menu.	ify how the ma	acro appears in	the		
- Menu					J -	
And Group				~		
Icon						
Show this ma	cro in the menu fo	r smartclient u	sers			
	cro in the menu fo					
Show this ma	cro in the menu fo	r centric users				
Keyboard						
	tcut key for this m	acro				
Key Alt+F	4					
		ок	Cancel	Help	-1	
i6				neip		

Movie Index

Conceptual	
What is RAMP? - 9 minutes	
5250 Application before using RAMP - 2 minutes	
5250 Application after using RAMP - 4 minutes	
Stage 1: Creating a Modernization Framework - 8 minutes	
Stage 2: Snapping the 5250 Application in the Framework - 11 minutes	
Stage 3: Enrichment and Re- engineering - 5 minutes	
Tutorial: Modernizing Complete App	lication
Application before Modernization - 2.5 minutes	Movie Summary
Modernized Application - 2 minutes	Movie Summary
Identify your business objects - 1 minute	Movie Summary
Create a prototype of your application - 3 minutes	Movie Summary
Create a filter and snap it in - 4 minutes	Movie Summary
Make a plan of the 5250 screens you will need to use - 2.5 minutes	Movie Summary
Identify the relevant screens and fields	Movie Summary

to newlook - 4 minutes	
Define the screens to the VLF and build a navigation script (New Employee) - 7.5 minutes	Movie Summary
Define the screens to the VLF and build a navigation script (Employee Details) - 5 minutes	Movie Summary
Link the Selected Employee in the Instance List with the Display Employee Screen - 4 minutes	Movie Summary
Make Function Keys Go Somewhere Different - 4.5 minutes	Movie Summary
Handle Unexpected Stops in Navigation and Messages - 3 minutes	Movie Summary
Update the Instance List from 5250 Screens - 4 minutes	Movie Summary
Tutorial: Modernizing Application Na	avigation
Modernized Navigation - 5 minutes	Movie Summary
Create a Prototype of Your Application - 9.5 minutes	Movie Summary
Identify Your 5250 Entry Point Screens Using newlook - 13 minutes	Movie Summary
Script the Screens and Snap them in the Framework - 16 minutes	Movie Summary
Scripting: General	
Introduction to Scripts - 6.5 minutes	Movie Summary

Reading, Writing and Storing Values in Scripts - 4 minutes	Movie Summary		
Debug and Diagnostics - 2.5 minutes	Movie Summary		
Scripting: Subfiles/Browselists			
Not Using a Datagrid Control - 1 minute	Movie Summary		
Using Subfile Accessor - 5 minutes	Movie Summary		
Subfile Direct Access - 2 minutes	Movie Summary		