

RAMP-TS Guide

Rapid Application Modernization Process for Terminal Server Guide

[How to Get Started with RAMP](#)

[What's New](#)

[Licensing Requirements](#)

[Installation and Configuration](#)

[Starting RAMP](#)

[Concepts](#)

[Modernization Issues](#)

[Tutorials for RAMP Windows](#)

[Scripting](#)

[Screen Enrichment](#)

[Screen Wrappers](#)

[Programming Techniques](#)

[Multilingual RAMP Applications](#)

[Troubleshooting](#)

[Frequently Asked Questions](#)

[Appendix](#)

Please send your comments and suggestions to LANSAs 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 LANSAs be liable for any damages arising from its use. LANSAs MAKES NO WARRANTIES, EXPRESSED OR IMPLIED.

Edition EPC130100

Edition Date November 14, 2012

© 2012 LANSA

How to Get Started with RAMP

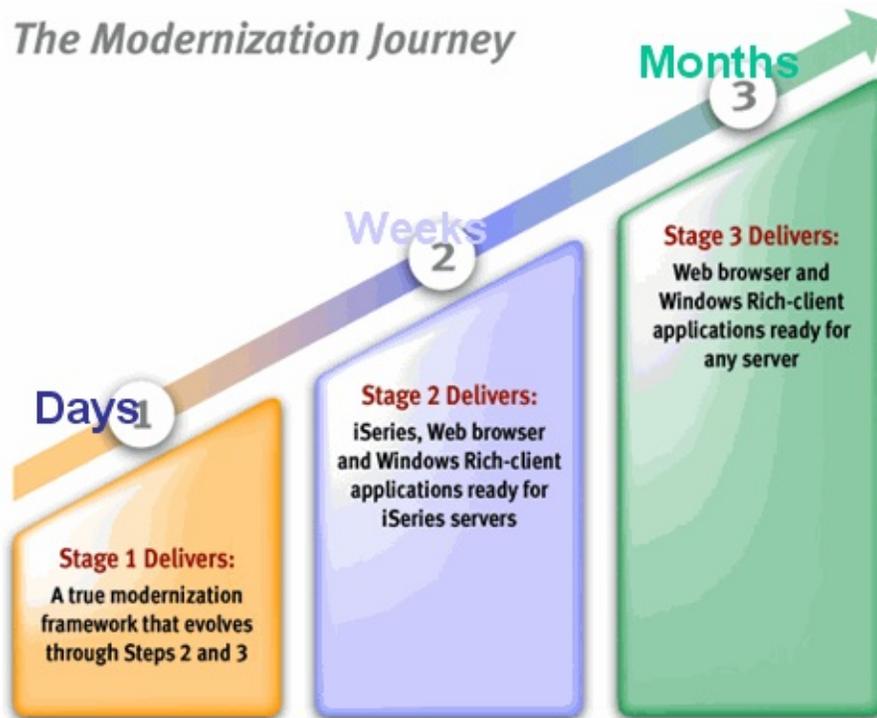
This is the recommended 5 step plan for getting started with RAMP:

1. Review the introductory section [What is RAMP?](#)
2. Comprehend how modernization will change the nature of a 5250 application by reviewing:
[5250 Application Before Modernization](#)
[5250 Application After Using RAMP](#)
3. Comprehend how the modernization process is performed by reviewing:
[Stage 1: Creating a Modernization Framework](#)
[Stage 2: Incrementally Modernizing the 5250 Application](#)
[Stage 3: Removing IBM i Platform Dependencies](#)
4. Install and Configure RAMP software (see [Installation and Configuration](#)).
5. Complete the essential [Tutorials for RAMP Windows](#).

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

What is RAMP?

RAMP is a staged process for the rapid modernization of IBM i (or) i5 based 5250 applications. It is an extension to the Visual LANSAs Framework.



[RAMP Stage 1](#) involves prototyping what your modernized application will look like when it is finished.

[RAMP Stage 2](#) involves re-animating the existing 5250 application in the Framework and incrementally modernizing it.

[RAMP Stage 3](#), which is optional, may involve you in re-engineering your application so as to remove any specific IBM i platform dependency that it has.

To understand what the end result will be, have a look at [A Modernized Application](#).

The RAMP process has been designed to support rapid and incremental deliveries of your modernized applications. It is not an approach where you have to work for years until the result comes out in one big bang. If you have

been involved in big bang projects before you will know of the pitfalls that this entails. This RAMP approach allows you to modernize at your own pace, thus allowing you to manage risk in a sensible manner. See [Key Benefits](#).

RAMP Stage 1



RAMP stage 1 creates a prototyped modernization framework.

The purpose of this step is to create an unencumbered vision of what "could be" rather than just "what is", in other words the prototype is produced to be a vision of the future rather than just a reflection of the application the way it is today.

This prototyping framework is important because:

- It defines where and how your 5250 screen panels will be reused
- It allows all stake-holders, be they developers or end users, to understand and review what they are going to get and what they are going to have to give to complete the modernization project.
- It is not thrown away, but will evolve into the real executable application framework.

You use the Instant Prototyping Wizard to create the prototype. This tool is especially useful for people with a 5250 background because it bridges the gap to Windows and web browser application design.

It should only take a few days to deliver a modernization framework for even a very large application. The prototyping process may be so rapid that multiple prototypes can be produced for comparison and review by the stakeholders.

At the end of this stage a real executable application framework is delivered. Not only does this ensure everyone knows what they are getting and what they have to do, it also makes it easy to define and manage the rest of the project and to assign the various deliverables to the project participants.

The application framework will be evolved into a real application by gradually snapping out the prototype

parts and snapping in real application parts which will be either reanimated 5250 screens or new functions.

For more information, see [Modernization Process Overview](#).

RAMP Stage 2



In RAMP stage 2 you snap in your 5250 application to the Framework and incrementally modernize it:

RAMP Stage 2A

In this step you rapidly modernize infrequently used complete application segments by simply snapping their entry screen (typically a menu) into the Framework. You should do this with all infrequently used segments of your application.

The major benefit of this approach is to give your 5250 screens a Windows user interface and to make them accessible from the Framework in very little time.

It is very important to understand that you do not have individually modernize every single 5250 screen in your application as in stage 2B. For more information, see [RAMP Stage 2A: Rapidly Modernizing Complete Application Segments](#).

RAMP Stage 2B

In this step you individually modernize specific application areas and screens. You use this approach with key parts of your application.

You record the navigation to the destination screens in the 5250 application and snap them in the Framework. You then create filters to provide the end-users with efficient access to the data.

The enhanced navigation and data access provide significant usability benefits and are the core of the modernized application. For more information, see [RAMP Stage 2B: Individually Modernizing Frequently Used Screens](#).

RAMP Stage 2C

In this step you can selectively enrich your existing 5250 application by adding new or enhanced functionality such as video, voice, e-mail, improved desktop integration, etc.

How much you add depends on how much time you have available. For more information, see [RAMP Stage 2C: Adding Value](#).

RAMP Stage 3



In RAMP stage 3, which is optional, you can create an application that can execute using server platforms other than the IBM i, for example Windows or Linux servers are possible.

If you are an ISV then your existing or new customers will probably welcome the new options that this opens up for them.

Your application can be generated into various 3GL languages to ensure maximum execution efficiency on the various server platforms such as .NET or WebSphere.

The re-engineering process may use any technology, for example you might choose to use an:

- SOA (Service Orientated Architecture) approach or
- MVC (Mode-View Controller) approach.

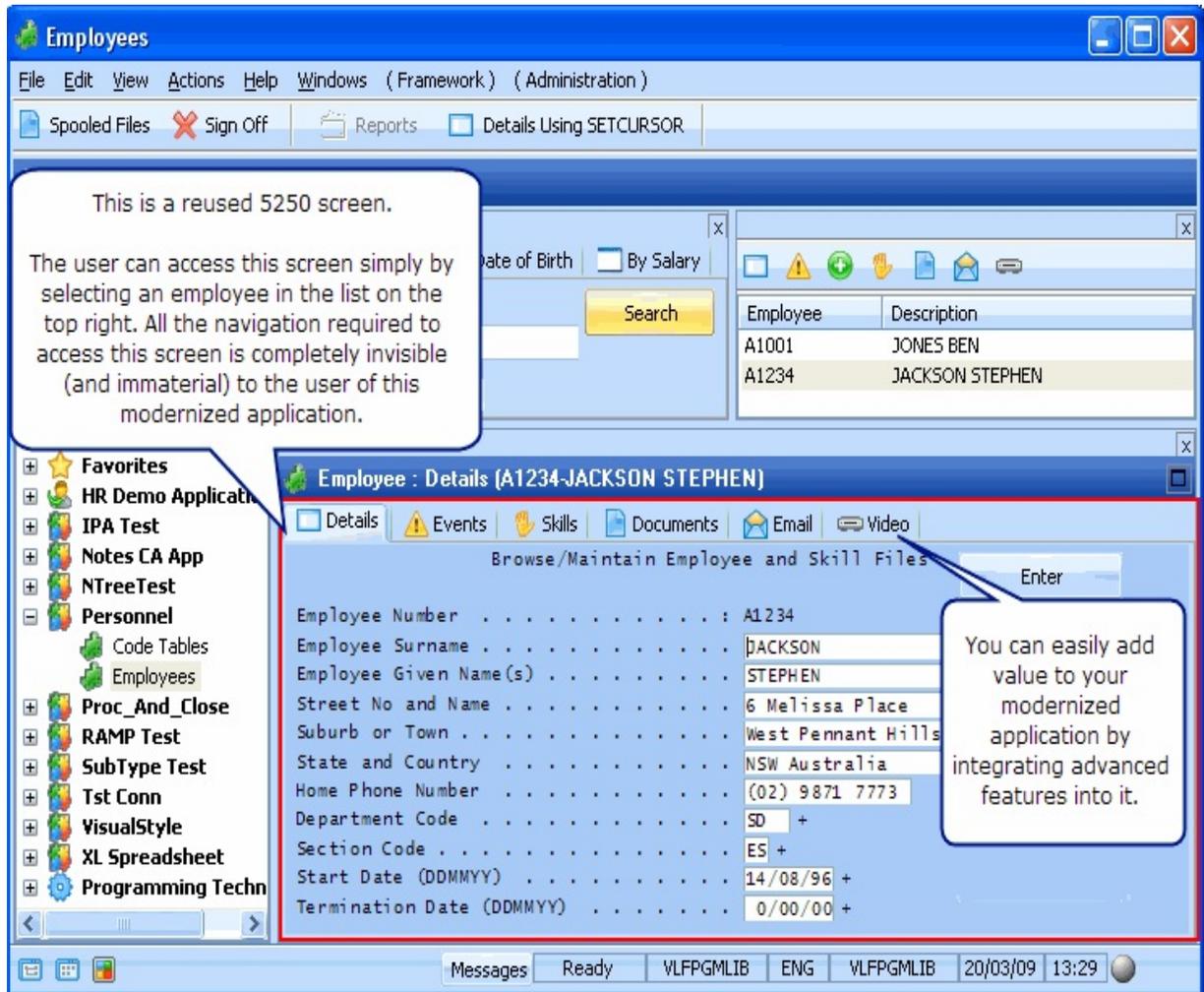
It's up to you.

This stage completely frees you from the limitations of the 5250 world.

For more information, see [Stage 3: Removing IBM i Platform Dependencies](#).

A Modernized Application

This is a 5250 Personnel application modernized by RAMP. It does not look anything like a 5250 application, but it is:



To move from, for example, the Details tab of an employee to the Skills tab the user simply clicks on the tab and the Skill 5250 screen appears. The fact that behind the scenes the 5250 application needs to press F12 twice, key in an employee number and then press enter is completely invisible to the user.

Notice tabs named Video and Email. Facilities like video and email are not things normally available in 5250 applications, but they can easily be added to a modernized application to enhance its functionality and usability.

Key Benefits

Uses one tool and one skill set	RAMP is part of LANSAs. One of LANSAs great strengths is that it allows developer to use a single tool and single skill set to solve all their commercial IT problems. A single developer with a single skill set can produce solutions ranging from Windows Rich Client applications, to Web Browser applications to creating an XML based document integration solution
Framework defines and drives modernization path	Uniquely RAMP uses a framework driven approach to application modernization. No one else does this. The framework defines and then manages the complete modernization process. It is the primary vehicle for managing expectations and communications among the various modernization project stakeholders.
Non-intrusive - no changes to existing 5250 applications	No change at all is required to your existing 5250 applications to use RAMP.
Supports all 5250 applications - even without DDS	RAMP can modernize any 5250 based application, even the application and operating system components that you have no DDS for (eg: Query, DFU and IBM i commands).
Not a "step sideways" or "ignore the problem" solution	RAMP provides an immediate short term solution (in stage 2) and then provides a clear path to move further forwards from there (in stage 3) to platform independence. Using RAMP is not a "step sideways" or "a band-aid" screen scraping solution that ignores your need to produce both a short term tactical solution for modernization and at the same time implement a longer term strategic direction for platform independence.
Incremental approach lowers	RAMP is not a big bang approach. It has been designed to accommodate rapid incremental deliveries of your modernized application. This minimizes your risk and time to market.

delivery time

Should not require hardware or O/S upgrades

RAMP will run on any platform that LANSAs supports. Currently LANSAs requires IBM i V5R1 or later. You, and if your are an ISV, your own customers, do not need to upgrade your hardware to support your RAMP modernized applications.

Delivers IBM i solution in short time

The entire focus of RAMP stage 2 is on the delivery of modernized IBM i applications in the shortest possible time. We understand that getting your application modernized and back out into the world in the shortest possible time is vitally important to you.

How the 5250 Application Will Change

The sample application shown in this section is part a 5250 ERP (Enterprise Resource Planning) application containing more than 3000 5250 screens.

It is a LANSAs 5250 application. However, RAMP is not limited to LANSAs applications and may be used on any IBM i 5250 application written in any language such as RPG or COBOL. RAMP may also be used on applications for which DDS does not exist such as DFU, Query and IBM i command 5250 screens.

No change to the 5250 application is made during the modernization process.

[5250 Application Before Modernization](#)

[5250 Application After Using RAMP](#)

5250 Application Before Modernization

Think for a moment of a 5250 or green screen application with classic navigation techniques in action: The users sign on and are greeted by a menu. They then navigate through several menus by selecting options until they are required to identify the item they want to work with:

```

Sign On
-----
System      . . . . . : S106E03E
Subsystem   . . . . . : QINTER
Display     . . . . . : QPADEV000C

User       . . . . . :
Password   . . . . . :
Program/proc . . . . . :
Menu       . . . . . :
Current library . . . . . :
  
```

The users sign on and are greeted by a classic 5250 menu. They select option 12 to display the Inventory Management Menu.

```

GARNET
-----
Garnet Main Menu
Enter number of function required or place cursor
12 1. Order Entry Menu
2. Customer Quote Menu
3. Shipping Menu
4. Billing Menu
5. Sales Analysis Menu
6. Accounts Receivable Menu
7. Accounts Payable Menu
8. General Ledger Menu
9. Currency Translation Menu
10. Cash Management Menu
11. Purchasing Menu
12. Inventory Management Menu
13. Bill of Material Menu
14. Cost Accounting Menu
15. Production Control Menu
16. Material Planning Menu

F1=Help F12=Return F14=Msgs
  
```

```

IMM001
-----
Item Maintenance Menu
Enter number of function required or place cursor on same line :
1 1. Item Master Maintenance
2. Alternate Item Master Maintenance
3. Plant Item Master Maintenance
4. Item Class Master Maintenance
5. Item Group Code Master Maintenance
6. Item Type Master Maintenance
7. Harmonizer Code Master Maintenance
8. Hazardous Material Code Master Maint
9. ABC Code Master Maintenance
10. Item Reference Maintenance
11. Plant, Whse and Location Maintenance
12. Other Inventory Maintenance
13. Work with Submitted Output
14. Work with Submitted Jobs
15. Display Messages
16. Return to Inventory Management Menu

F1=Help F12=Return F14=Msgs
  
```

```

IMM001
-----
Item Maintenance Menu
Enter number of function required or place cursor on same line :
1
IMUIMF -01 Item Master
Item Number A100BALL
Note: No audit trail will be produced.
F1=Help F3=Exit F4=Prompt F6=Add/Create F12=Return F14=Msgs
13. Work with Printed Output
14. Work with Submitted Jobs
15. Display Messages
16. Return to Inventory Management Menu

F1=Help F12=Return F14=Msgs
  
```

```

IMUIMF -03 Item Master 1/26/06 16:55:51
Maintenance
Item Number A100BALL
Action: 2-Change 3-Copy 4-Delete 5-Display 7-Notes
12-Work With 13-Plant/Item 15-Multilingual Descriptions
Act Item Number Description
2 A100BALLS Infiniti Golf Balls
A100CAP Infiniti Golf Cap
A100CAPBOB Infiniti Golf Cap with Bob Logo
A100COVER Infiniti Head Covers
A100BB Infiniti 9" Staff Bag
A100PLAQUE Infiniti Golf Plaque
A100SHIRT-L Infiniti Golf Shirt Large
A100SHIRT-M Infiniti Golf Shirt Medium
A100SHIRT-S Infiniti Golf Shirt Small
A100SHIRT-XL Infiniti Golf Shirt Extra Large
A100SHIRT-XXL Infiniti Golf Shirt Extra-Extra Large
A100V1 Infiniti Golf Balls
A200BB Infiniti 8" Carry Bag
B1A 234427 AGUA DE TOCADOR VAPOR L.BIAGIATTI ROMA

F1=Help F3=Exit F4=Prompt F5=Refresh F6=Add/Create F12=Return F14=Msgs
F16=Reseq
  
```

```

IMUIMF -04 Item Master 1/26/06 16:55:57
Maintenance
Screen Mode CHANGE
Item Number A100BALLS
Description Infiniti Golf Balls
Extra Descr Extra Description test
Short Descr
Item Class CLASS Item Type PUR Item Group Code ACCESS
Item Division
Tax Class
Manufactured? 0 (0/1) Non-Stock Item? 0 (0/1)
Group Tech Code
Buyer
Supplier ID
Rebates Flag (0/1) Item Status 1 Omit from Web
Ship-From Plant
Unit of Measure EA Dft Purchase UM SL Dft Sales UM DZ
Pricing U/M DZ Itm Catalog No
Res. Time Fence
Univ. Prod Code
Drawing/Formula
F1=Help F3=Exit F4=Prompt F12=Return F14=Msgs
  
```


5250 Application After Using RAMP

Here is the same 5250 ERP application as a Windows rich client application after RAMP has been used over it:

The screenshot displays the 'Inventory Objects' application window. The interface includes a menu bar (File, Edit, View, Actions, Tools, Help), a toolbar with 'New', 'Sign Off', and 'Details' buttons, and a tree view on the left showing the application structure. The main area is divided into a search section and a details section.

Search Section: The 'Find by Key' tab is active, with 'by Item Code' selected. The 'Item Number' field contains 'A100'. A 'Search' button is visible. Below the search results, the selected item is 'A100BALLS - Infnit Golf Balls'.

Item	Description
A100BALLS	A100BALLS - Infnit Golf Balls
A100CAP	A100CAP - Infnit Golf Cap
A100CAPBOB	A100CAPBOB - Infnit Golf Cap with Bob Logo
A100COVER	A100COVER - Infnit Head Covers
A100GB	A100GB - Infnit 9" Staff Bag
A100PLAQUE	A100PLAQUE - Infnit Golf Plaque
A100SHIRT-L	A100SHIRT-L - Infnit Golf Shirt Large

Details Section: The 'Item Edit' window is open for 'A100BALLS - Infnit Golf Balls'. The 'Screen Mode' is 'CHANGE'. The 'Item Number' is 'A100BALLS' and the 'Description' is 'Infnit Golf Balls'. The 'Extra Descr' field contains 'Extra Description test'. The 'Item Class' is 'CLASS', 'Item Type' is 'PUR', and 'Item Group Code' is empty. Other fields include 'Item Division', 'Tax Class', 'Manufactured?' (0), 'Non-Stock Item?' (0), 'Group Tech Code', 'Buyer', 'Supplier ID', 'Rebates Flag', 'Ship-From Plant', 'Unit of Measure' (EA), 'Dft Purchase UM' (SL), 'Dft Sales UM' (DZ), 'Pricing U/M' (DZ), 'Res. Time Fence', and 'Univ Prod Code'.

Annotations:

- The user has searched inventory items based on inventory number using a Framework filter.
- The filter has built a list of inventory items that match the search criteria. The end-user selects an item in the list to work with.
- The 5250 menu system has been replaced by an application tree which is used to navigate between different parts of the application.
- The snapped-in 5250 screen is used to maintain the details of the inventory item.

Accessing Data in the Modernized Application

Framework filters provide one of the most powerful ways of adding value to a modernized application because they offer the end-users efficient and flexible access to the data in the application.

The filter allows the user to build lists of inventory items.

The filter in this application allows the user to build lists of inventory items in more than 20 different ways. For example, the filter allows the end-user to locate inventory items by Item Code, Buyer, Shipment etc.

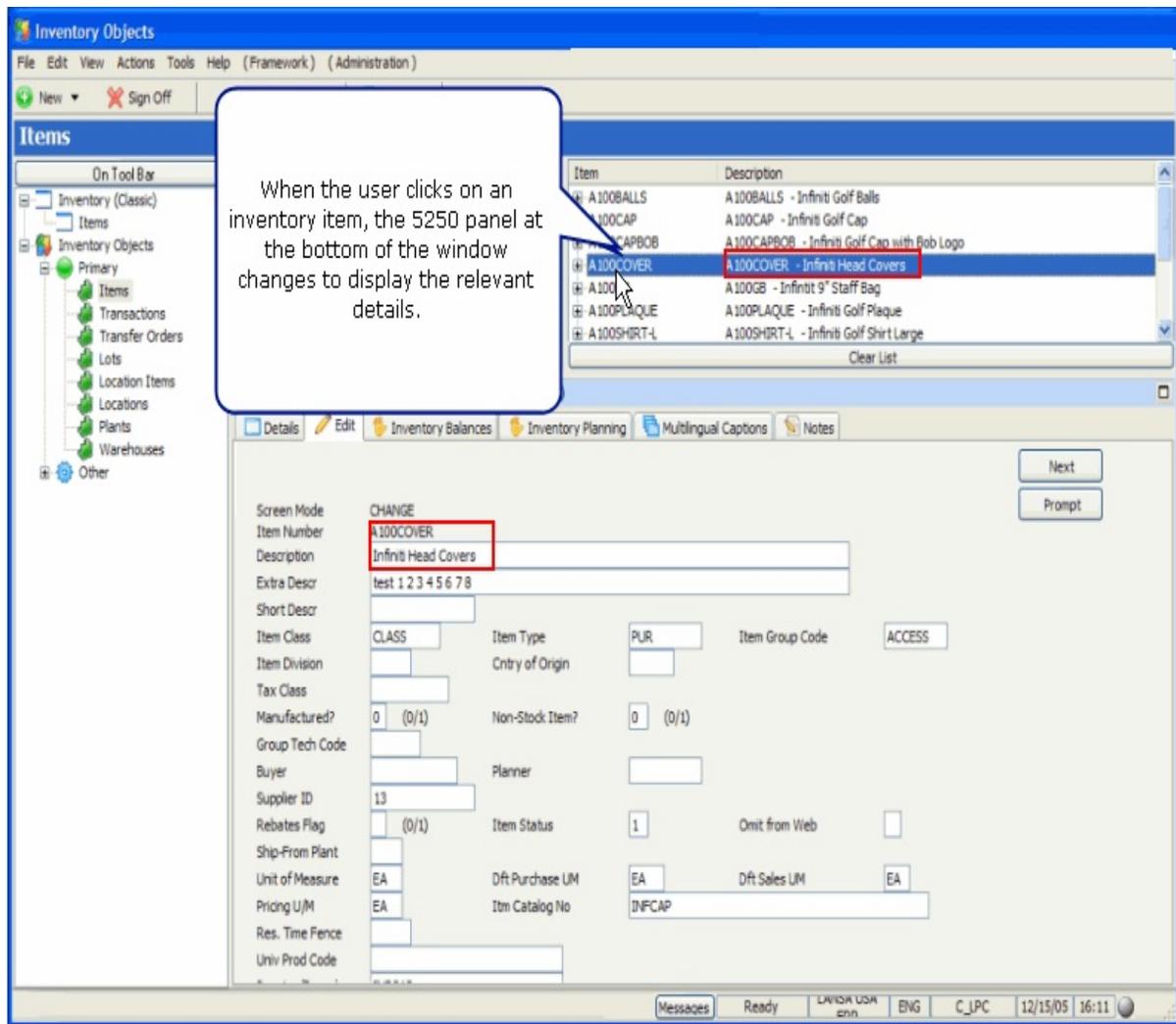
The filter builds a list of inventory items which is displayed as a tree. The user can drill down through it or sort it by any of the displayed columns. To work with an inventory item, the user simply clicks on the item in the list.

Item	Description
(+) A100BALLS	A100BALLS - Infnit Golf Balls
(+) A100CAP	A100CAP - Infnit Golf Cap
(+) A100CAPBOB	A100CAPBOB - Infnit Golf Cap with Bob Logo
(+) A100COVER	A100COVER - Infnit Head Covers
(+) A100GB	A100GB - Infnit 9" Staff Bag
(+) A100PLAQUE	A100PLAQUE - Infnit Golf Plaque
(+) A100SHIRT-L	A100SHIRT-L - Infnit Golf Shirt Large
(+) A100SHIRT-M	A100SHIRT-M - Infnit Golf Shirt Medium
(+) A100SHIRT-S	A100SHIRT-S - Infnit Golf Shirt Small
(+) A100SHIRT-XL	A100SHIRT-XL - Infnit Golf Shirt Extra Large
(+) A100SHIRT-XXL	A100SHIRT-XXL - Infnit Golf Shirt Extra-Extra L
(+) A100V1	A100V1 - Infnit Golf Balls

Navigation in the Modernized Application

In RAMP applications the Windows interface is driving the 5250 panels, rather than the 5250 screens driving the Windows application.

The difference in who is driving - the Windows application or the 5250 application - is what makes RAMP different to, and so much more powerful than, any traditional form of 5250 screen scraping technology.



Inventory Objects

File Edit View Actions Tools Help (Framework) (Administration)

New Sign Off Details Calculator

Item

If the users want to see the Notes associated with the selected inventory item, they click on the Notes tab and the 5250 screen showing the notes appears.

In an unmodernized 5250 application they would have had to use F12 twice, chosen an item from a menu and then entered the item number again to accomplish this.

Notes

Item Number: A100COVER Infniti Head Covers

Attention Code:

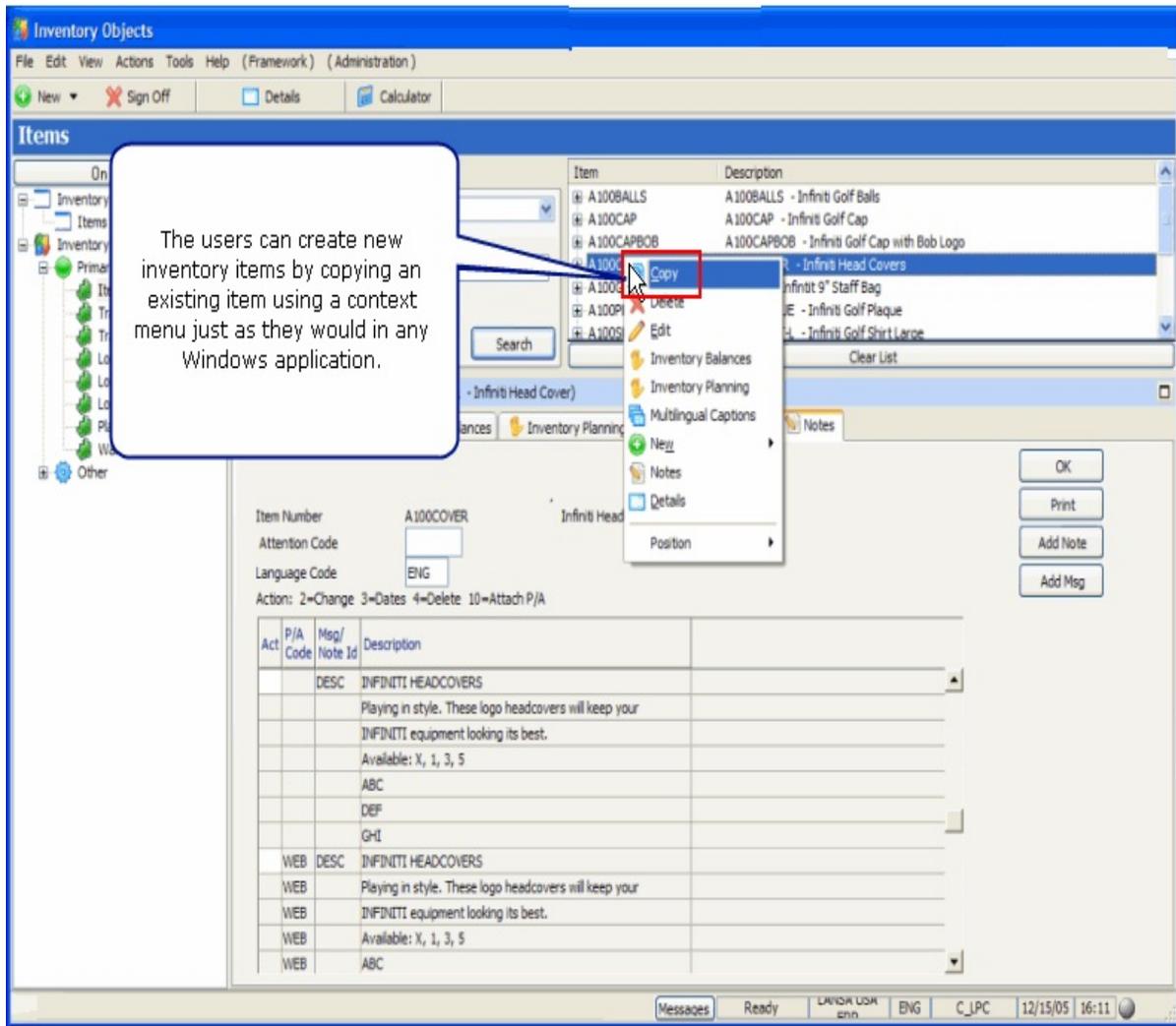
Language Code: ENG

Action: 2=Change 3=Dates 4=Delete 10=Attach P/A

Act	P/A Code	Msg/Note Id	Description
	DESC		INFINTI HEADCOVERS
			Playing in style. These logo headcovers will keep your INFINTI equipment looking its best.
			Available: X, 1, 3, 5
			ABC
			DEF
			GHI
WEB	DESC		INFINTI HEADCOVERS
WEB			Playing in style. These logo headcovers will keep your
WEB			INFINTI equipment looking its best.
WEB			Available: X, 1, 3, 5
WEB			ABC

OK Print Add Note Add Msg

Messages Ready LANUSA ENG C_LPC 12/15/05 16:11



RAMPed applications do not just look better, they are easier and more productive to use and move around in, and easier to teach someone new to use. You can easily extend the modernized application by adding in new features such as e-mail, voice, video, web browser interfaces, desktop integration features, etc. that are not possible using the 5250 interface.

Modernization Process Overview

Stage 1: Creating a Modernization Framework

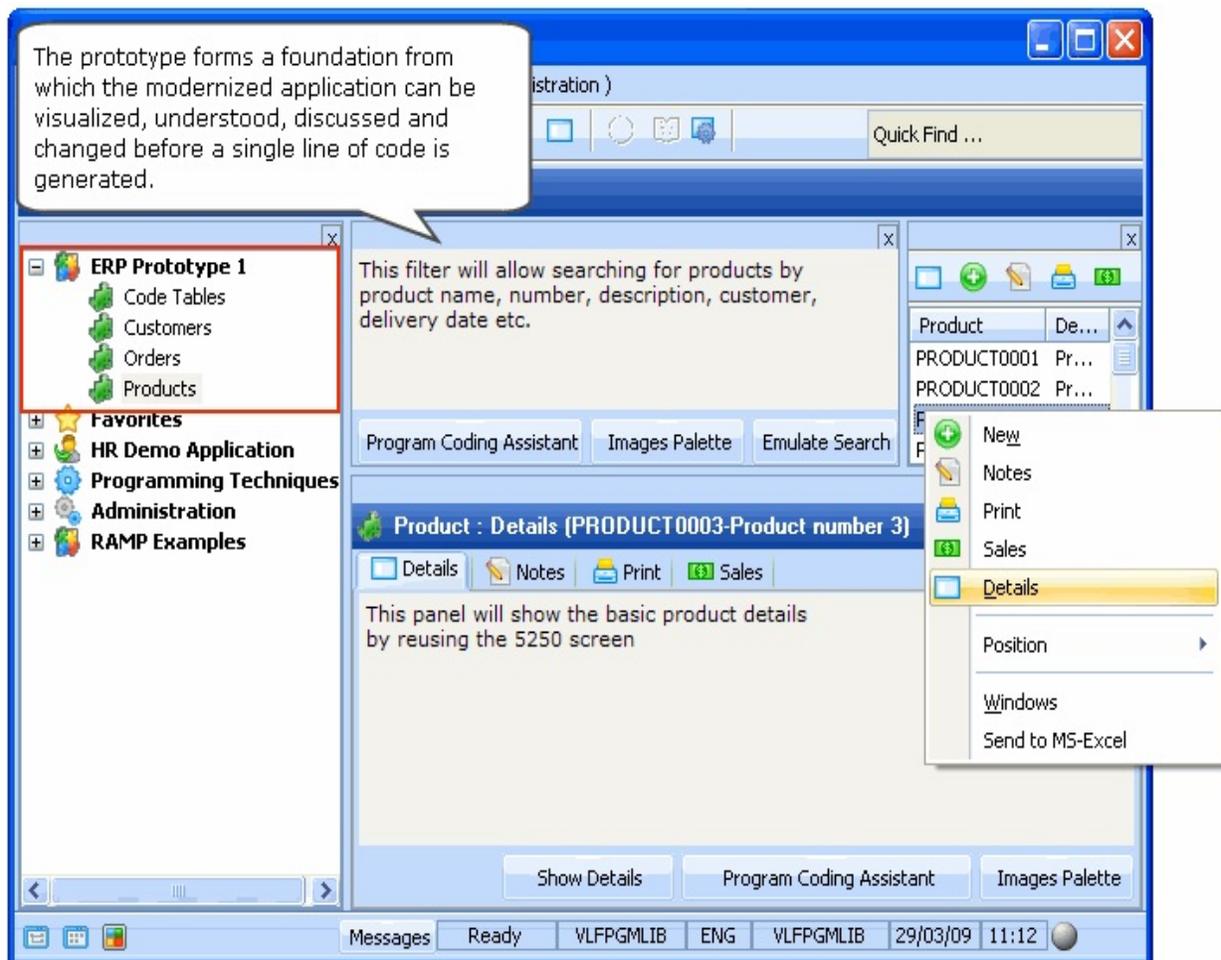
Stage 2: Incrementally Modernizing the 5250 Application

Stage 3: Removing IBM i Platform Dependencies

Stage 1: Creating a Modernization Framework

To use RAMP the first step is always to produce an executable prototype of what your modernized application will be like when it is completed. The prototype is very important because it helps to make sure that all stake-holders, be they developers or end-users, understand what they are going to get and what they are going to have to give before any significant investment is made in the result.

Fortunately RAMP makes application prototyping very simple and rapid.



Who Should Be Involved?

Creating the Prototype

Executing and Refining the Prototype

Who Should Be Involved?

Prototyping requires effective communication between management, users and IT.

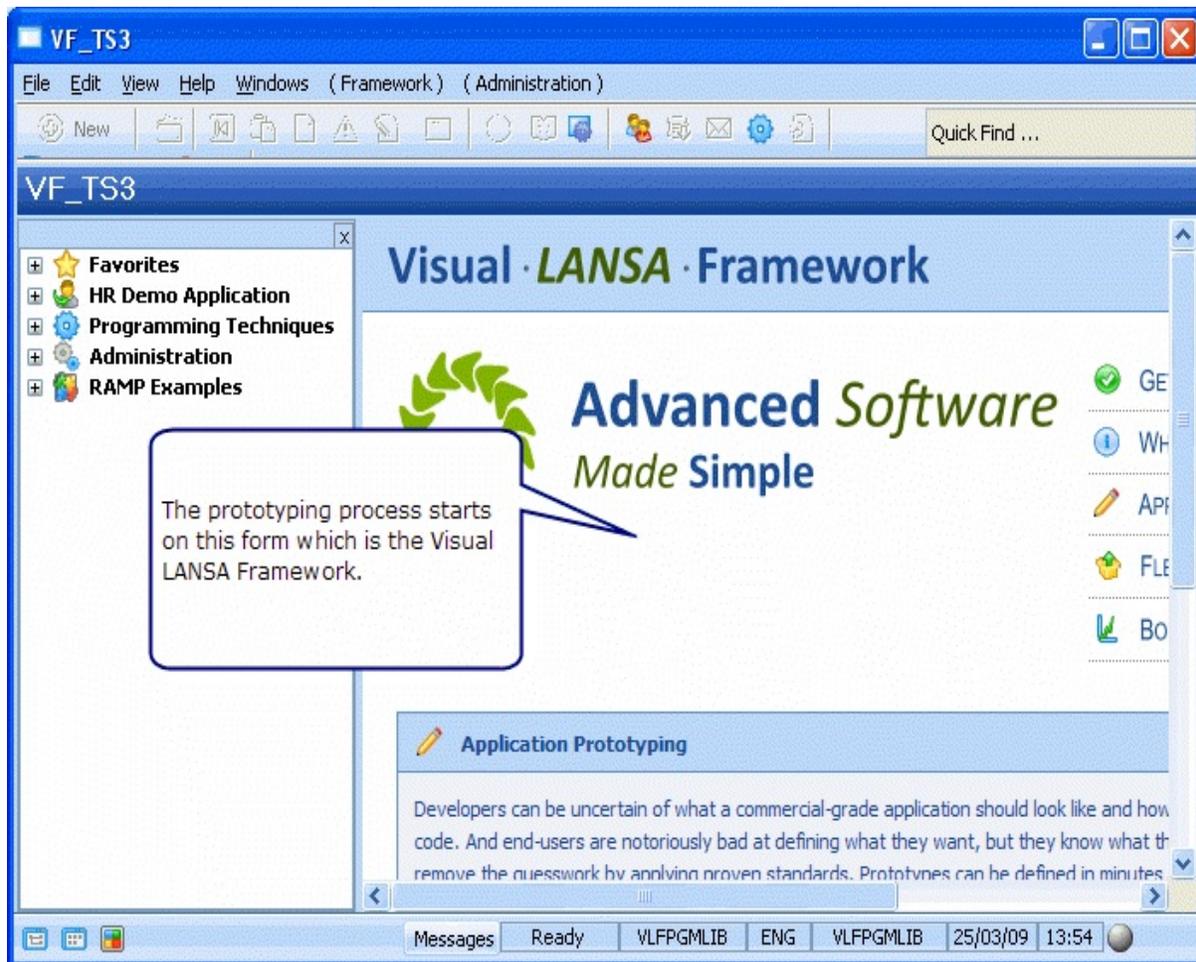
The classic idea of having management and user project champions still holds strongly. Without project champions IT groups will always struggle with identifying and adding real value to any project.

Asking the project champions to identify and then design the top five (say) most important real value additions, then fully implementing them, will sometimes get their complete commitment to a project and reinforce the benefits of thinking clearly, as a group, about real value, as opposed to getting lost in a maze of trivial value items like radio buttons, drop downs and trees.

At this stage you should review the topic [What Adds Value?](#)

Creating the Prototype

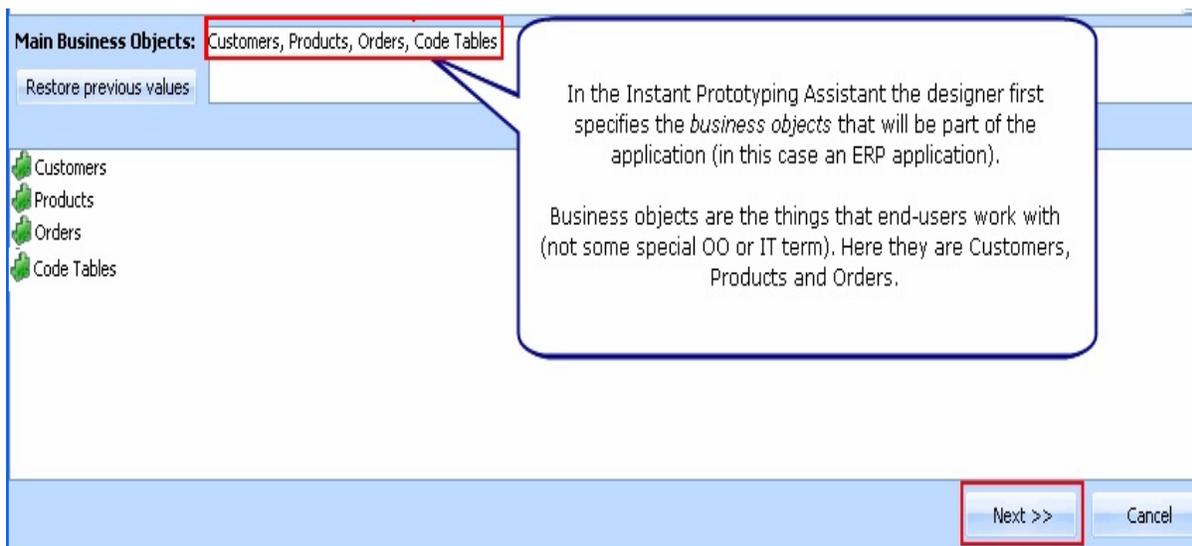
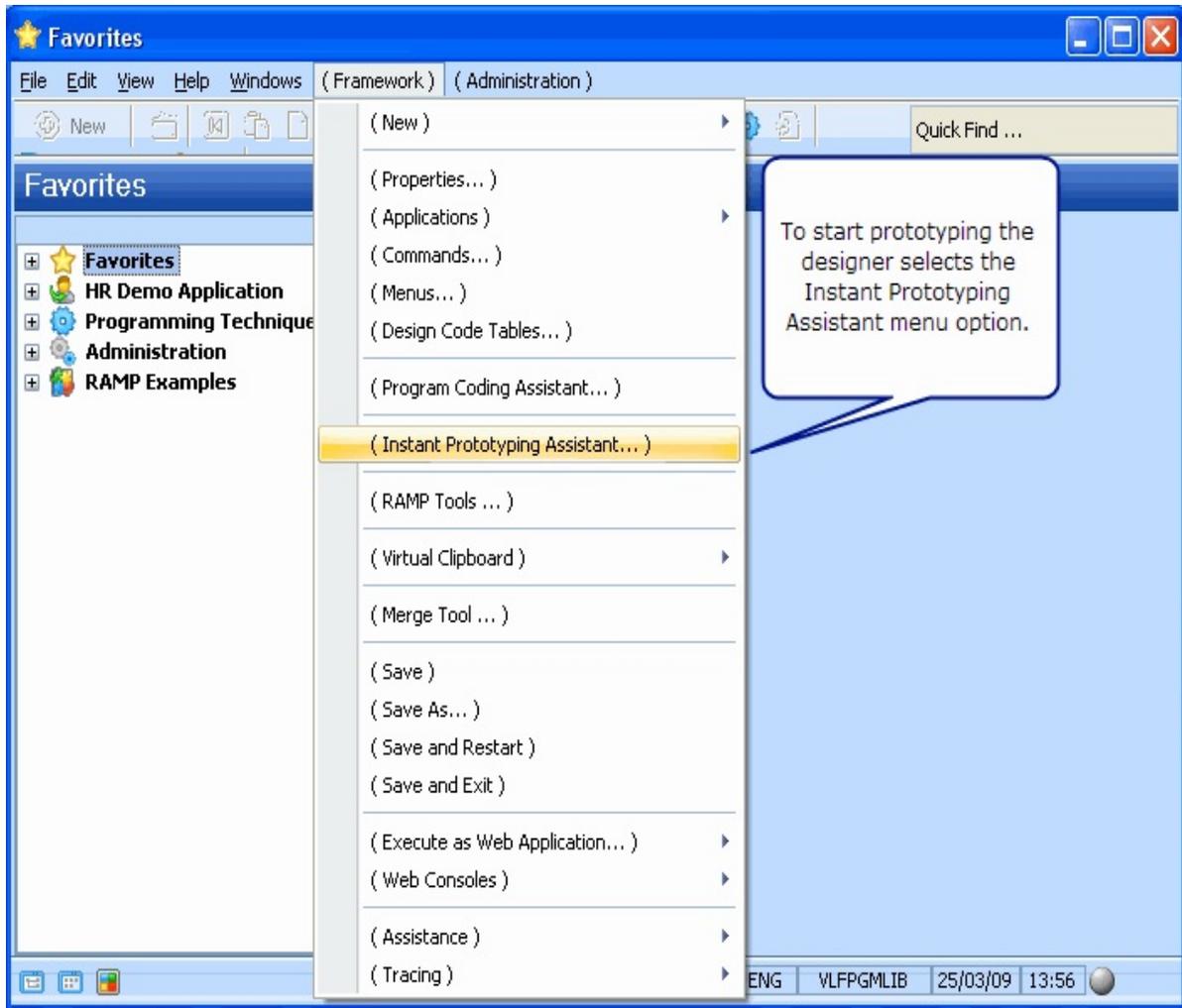
The prototyping process is so rapid that it is feasible to produce multiple prototypes that may be compared and discussed.



The Visual LANSA Framework is a framework for defining and executing commercial applications.

The main window is laid out in a MS-Outlook or "dashboard" style. For more information refer to [Key Concepts](#) in the Framework Guide.

Where these various form areas are and how they are laid out can be changed in many ways by the designer and even the end user according to individual preference. See [Tailoring the Window Layout](#) in the Framework Guide.



Actions: Details , New , Notes , History, Sales, Contacts, Print, Approval

Step 3. Drag and drop the actions from the list on the left onto the list on the right

The same action can be used with many business objects.

Next the Assistant asks what *actions* will the end-users be taking against these business objects.

There are predefined actions and the designer adds some more, so the final list is: Details, New, Notes, History, Sales, Contacts, Print and Approval.

- Details
- New
- Notes
- History
- Sales
- Contacts
- Print
- Approval

... (exists already)
 ... (exists already)
 ... (exists already)
 ... (exists already)

<< Back Next >> Cancel

Actions: Details , New , Notes , History, Sales, Contacts, Print, Approval

Step 3. Drag and drop the actions from the list below, onto all the

The same action can be used with many business objects.

Then the designer associates the actions with the business objects by drag-and-drop.

- Details
- New
- Notes
- History
- Sales
- Contacts
- Print
- Approval

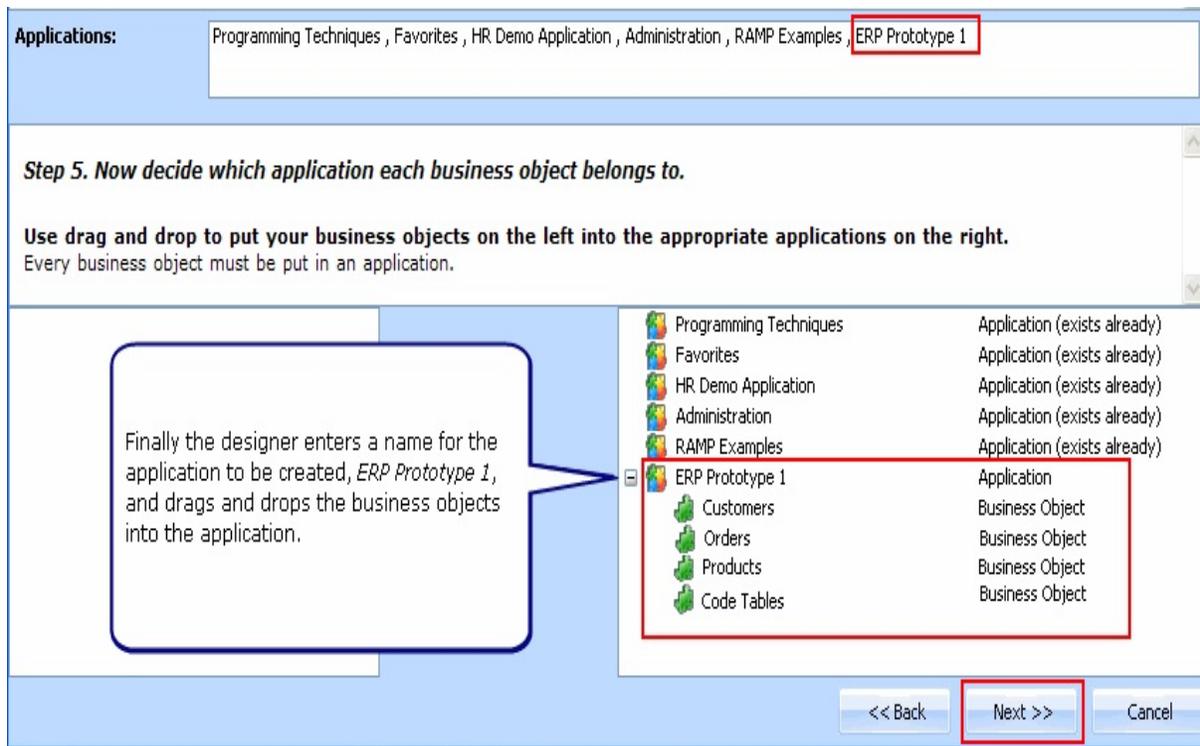
Customers

- Details Action: Details Customer
- New Action: New Customer
- Notes Action: Notes Customer
- History Action: History Customer

Products

- Details Action: Details Product
- New Action: New Product
- Notes Action: Notes Product

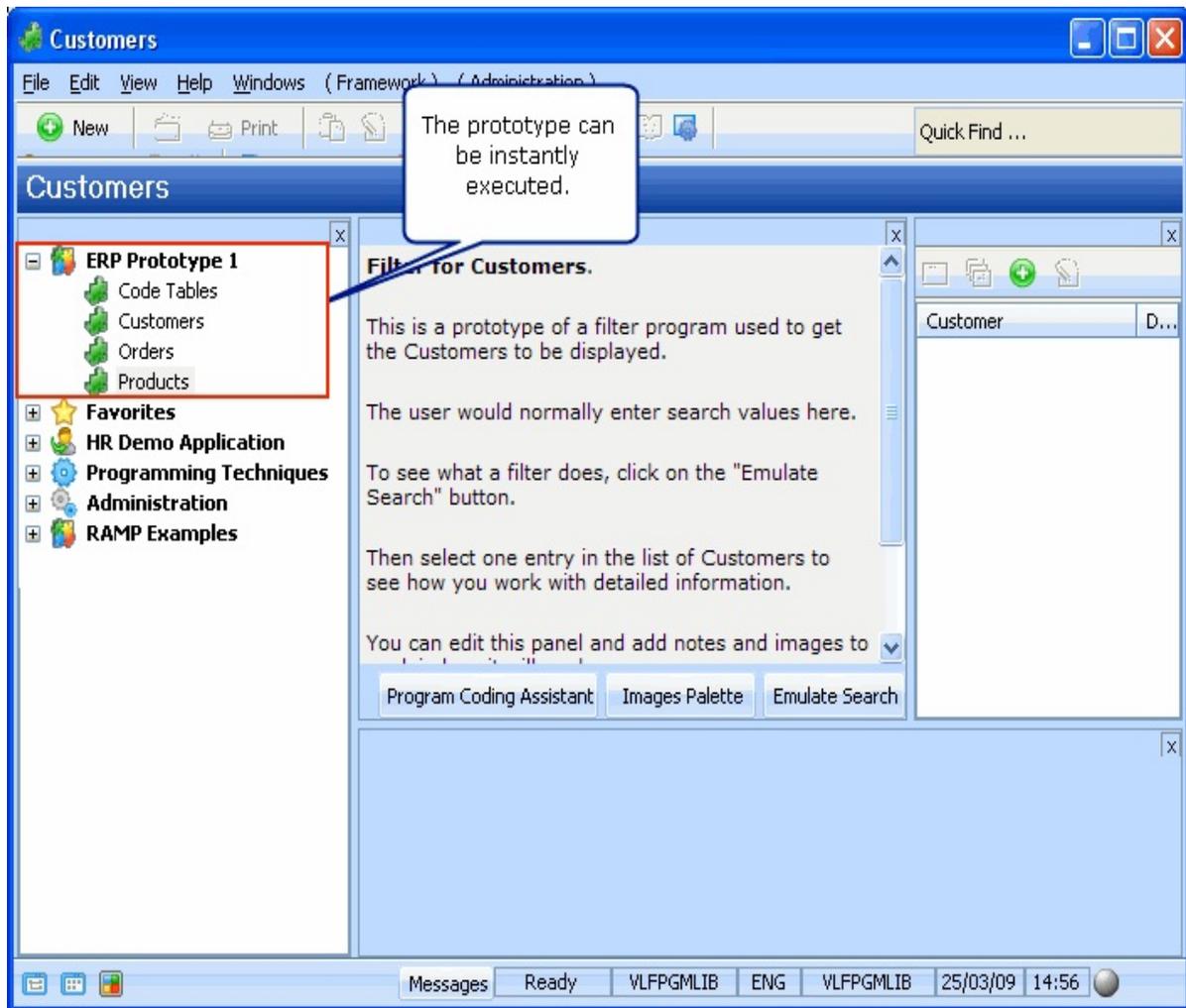
<< Back **Next >>** Cancel

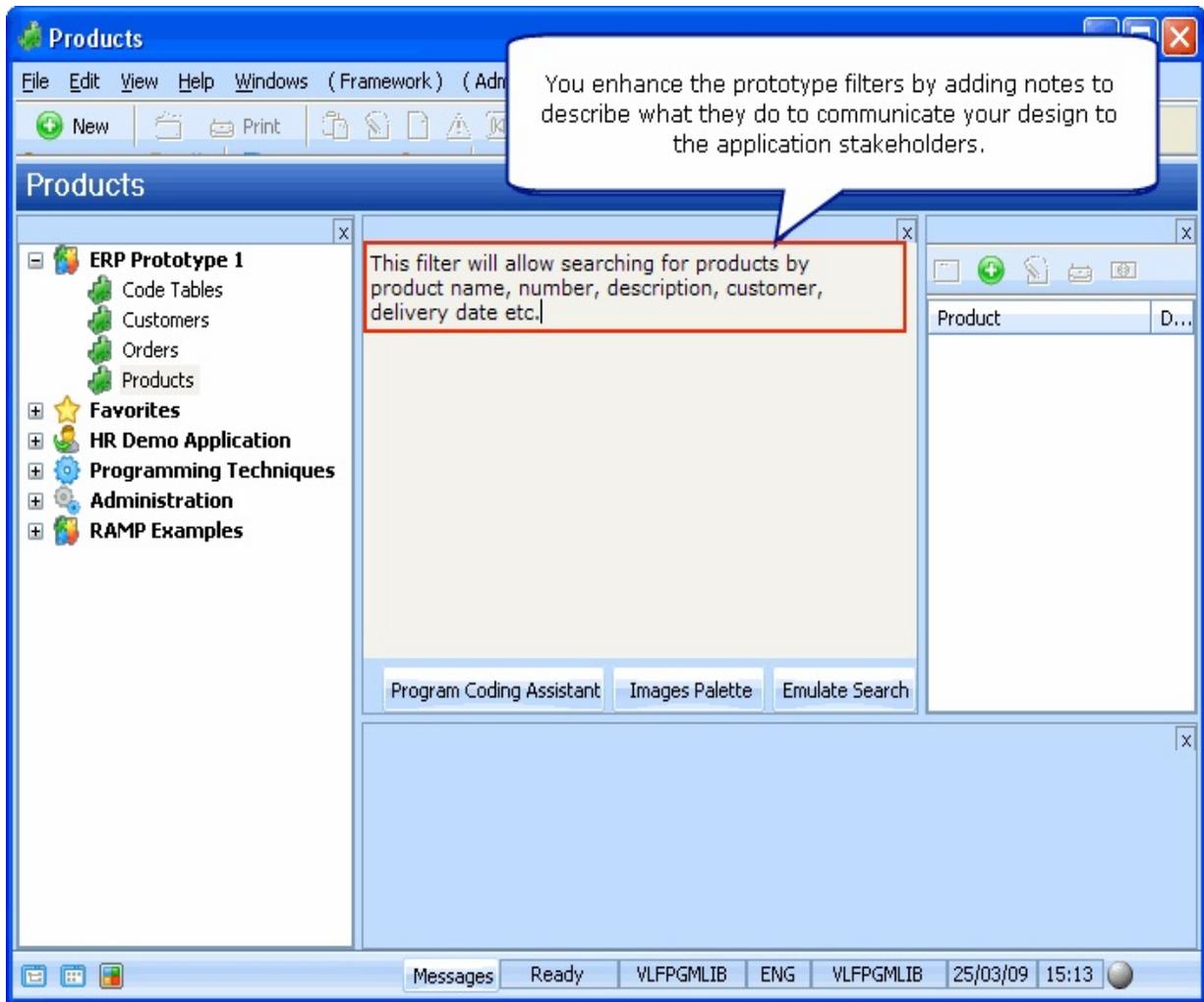


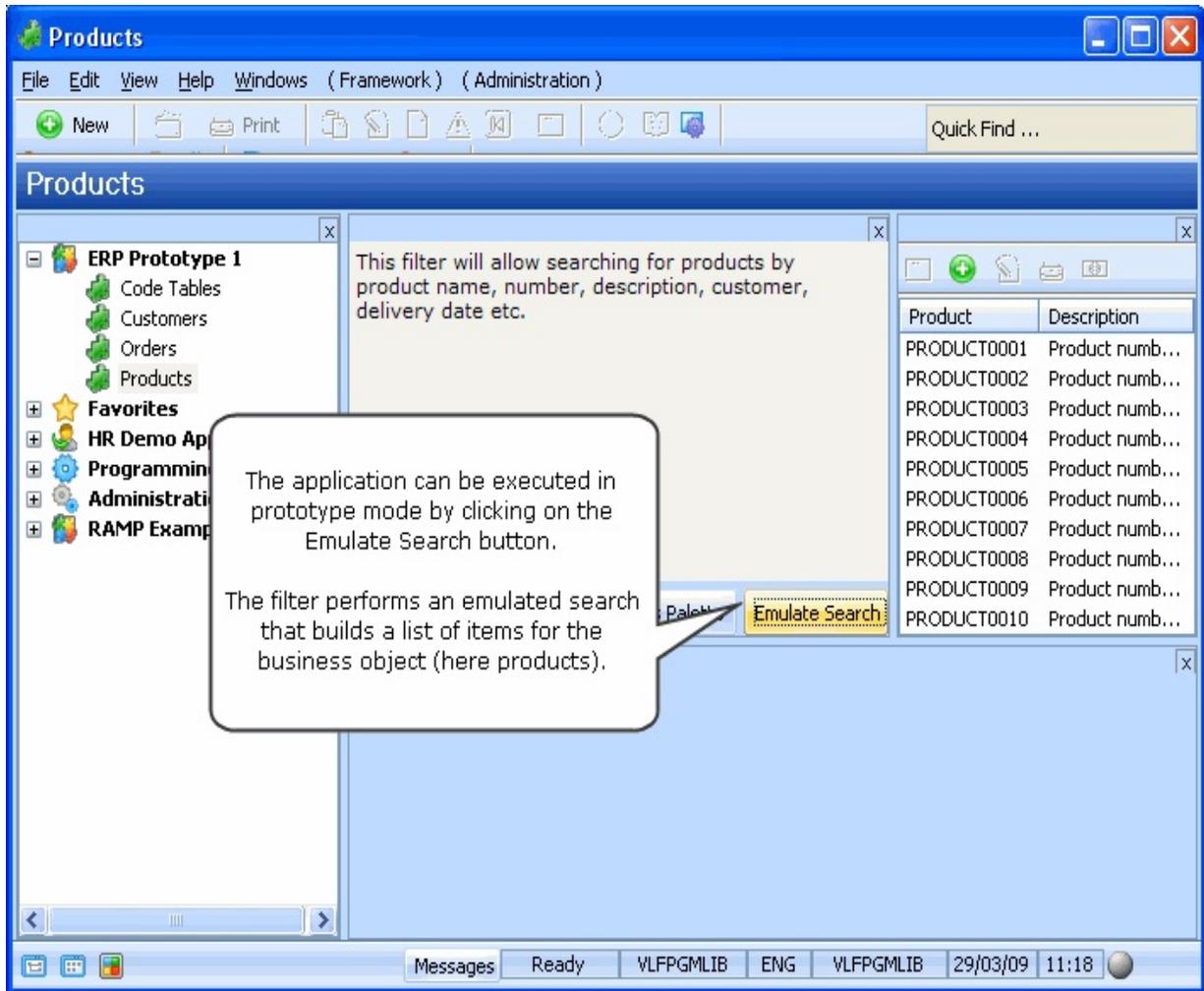
The user then clicks on the Finish button and the prototype application is generated. The Instant Prototyping Assistant closes and the Visual LANSA Framework main windows reappears.

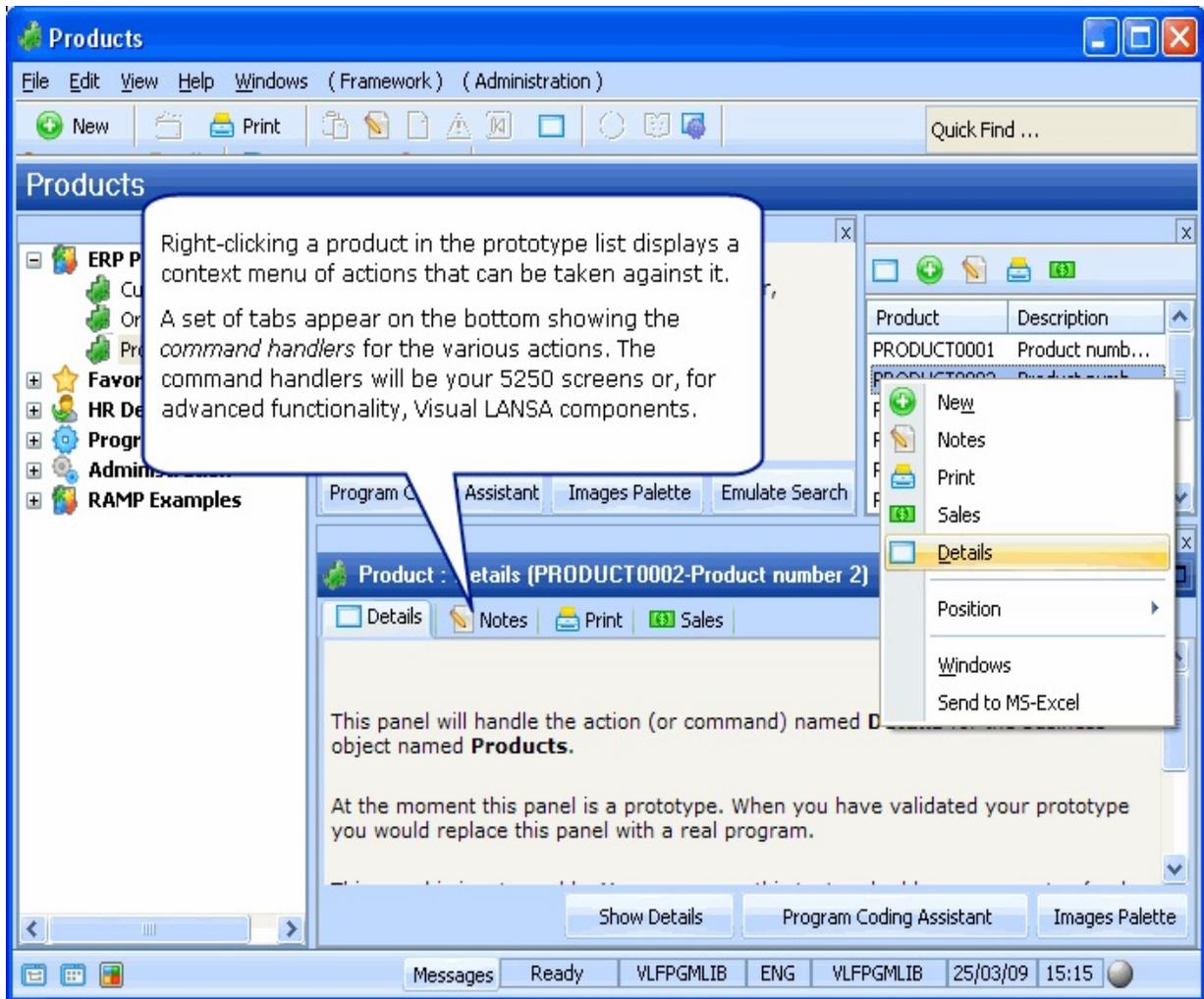
Executing and Refining the Prototype

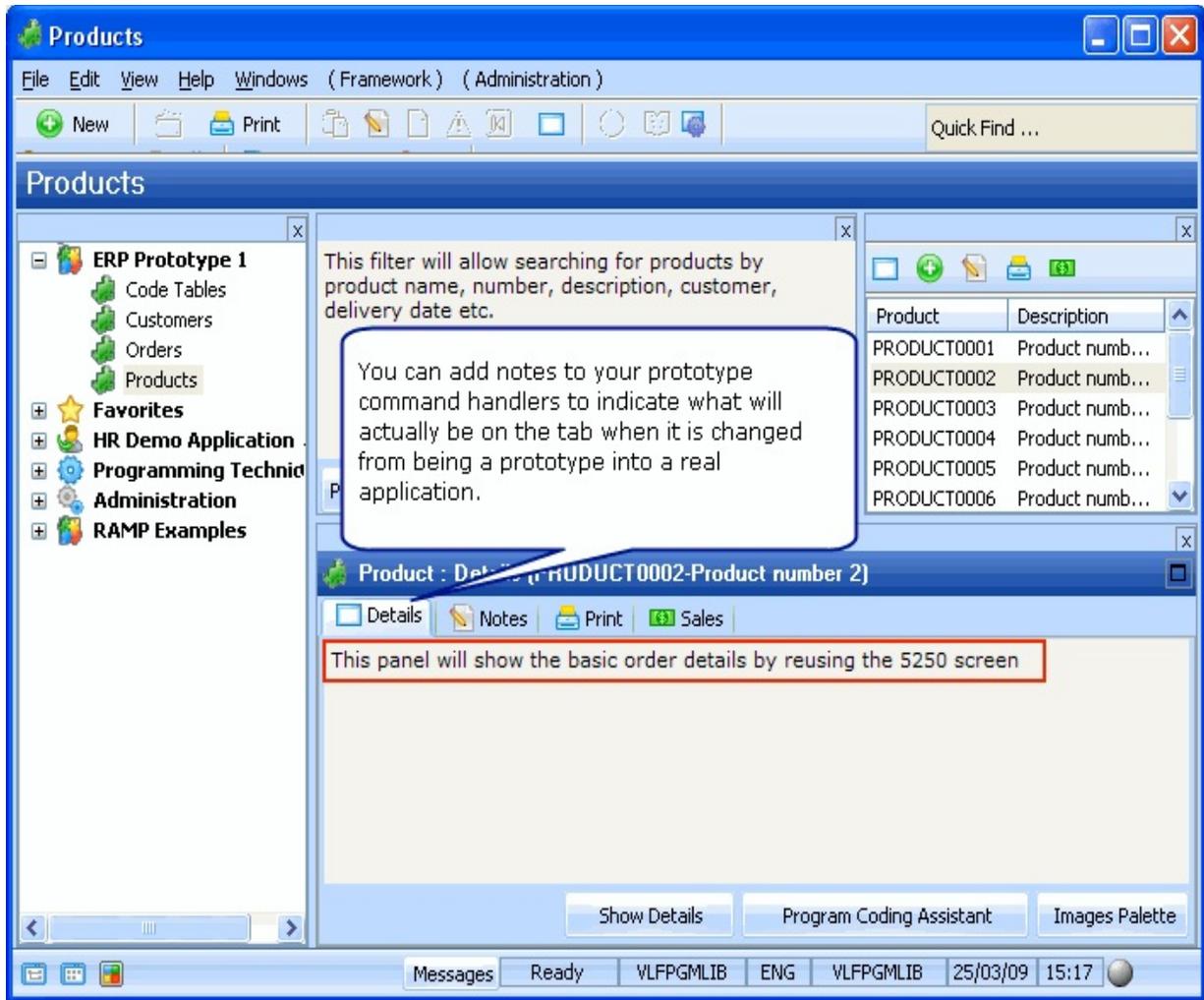
The prototype ERP application has now been created and inserted into the framework.

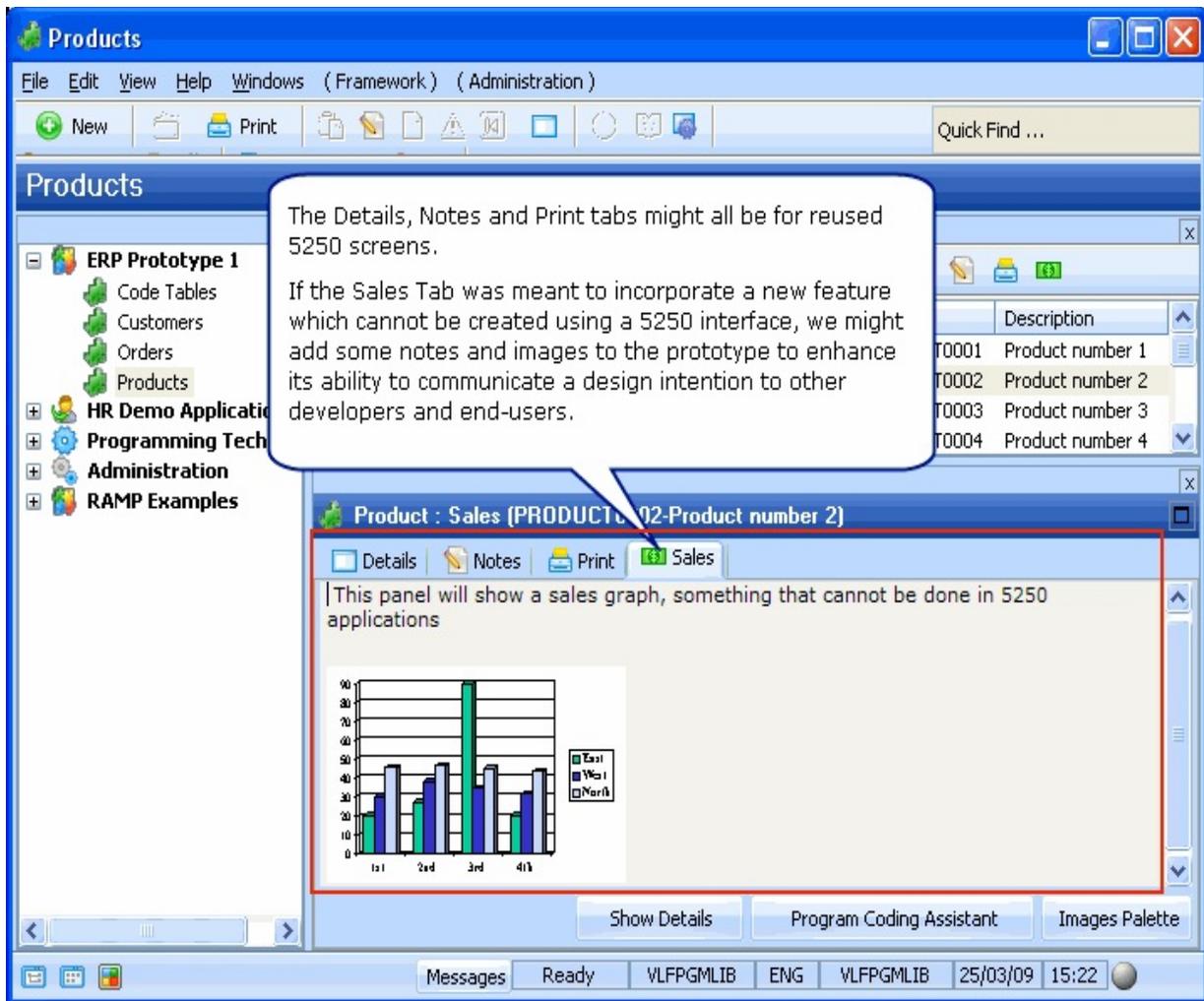




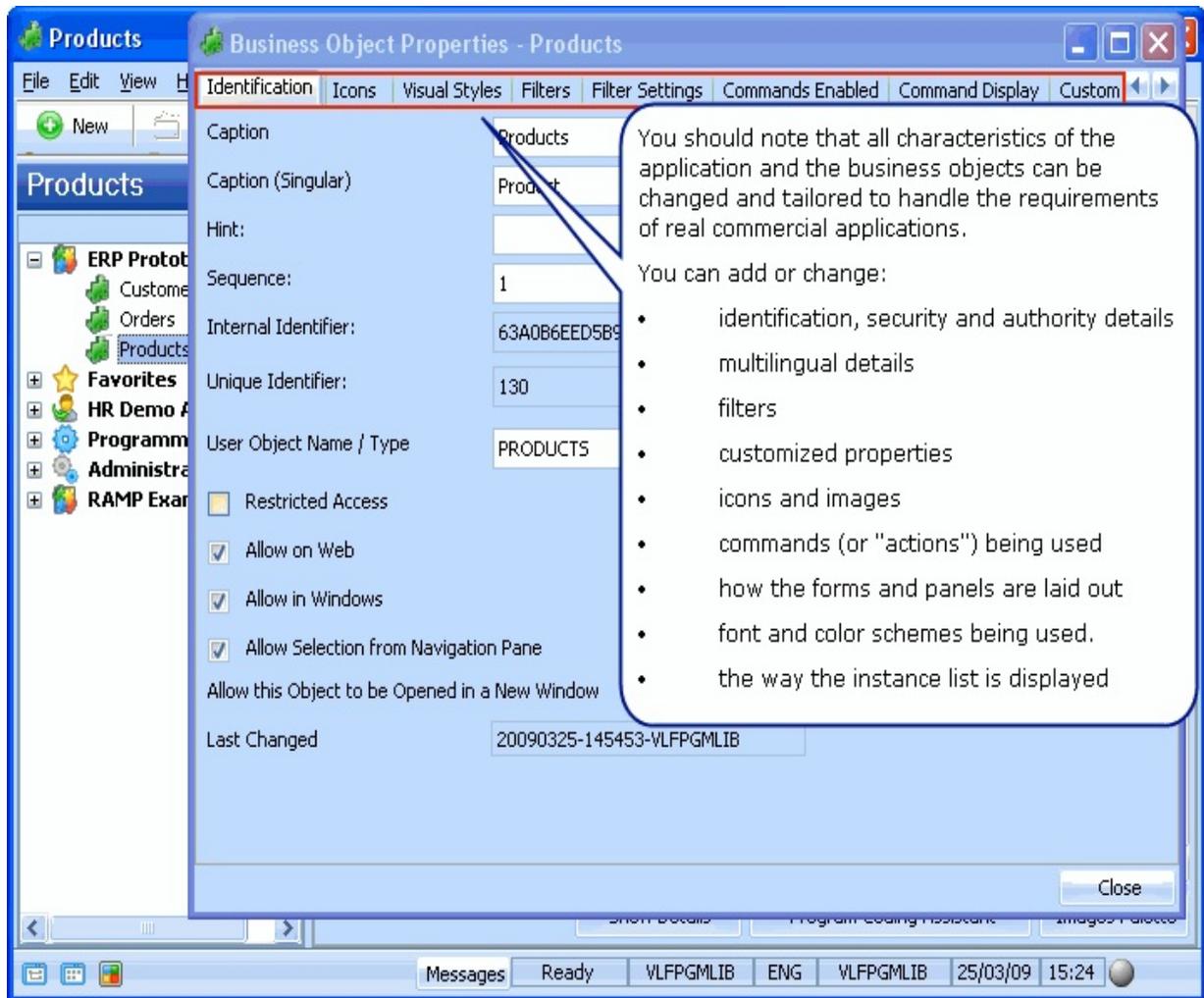








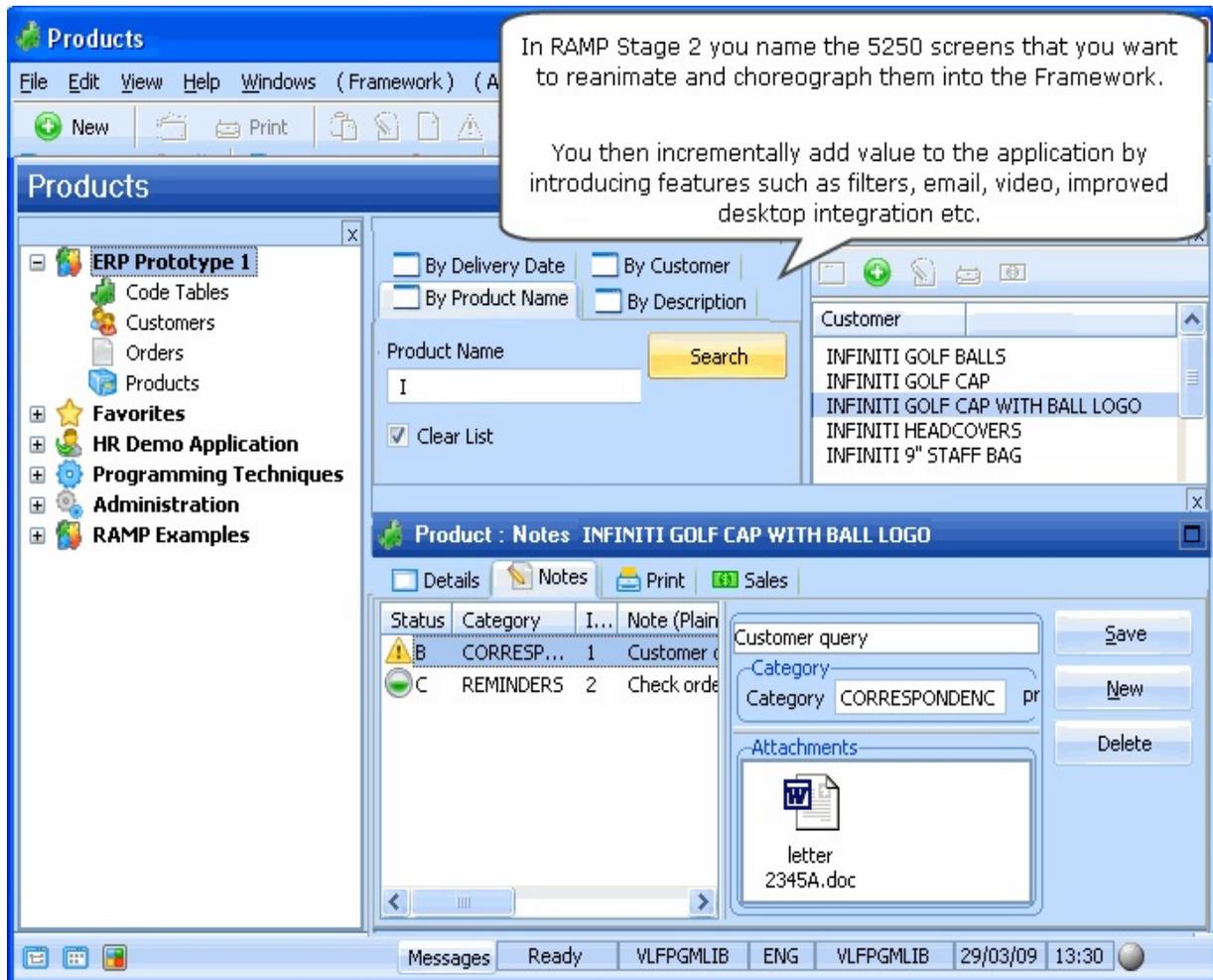
When you are happy with the prototype application, you can proceed to implement the various parts of it as a real application, gradually snapping out the prototype parts and snapping in the real application parts, so over time the prototype evolves into a real application.



For detailed instructions for how to prototype an application, refer to the tutorial [RAMP-TS001: Creating a Basic Prototype of the Modernized Application.](#)

Stage 2: Incrementally Modernizing the 5250 Application

RAMP has been designed to provide a realistic way to manage risk and time to market issues by using an incremental and evolutionary approach to producing the next generation of an application. In other words, because the RAMP architecture is not a “big bang” approach, it gives the developer the choice of shipping multiple small incremental versions of their application to minimize risk and the time to market.



Naming the 5250 Screens

Tracking and Classifying the Screens

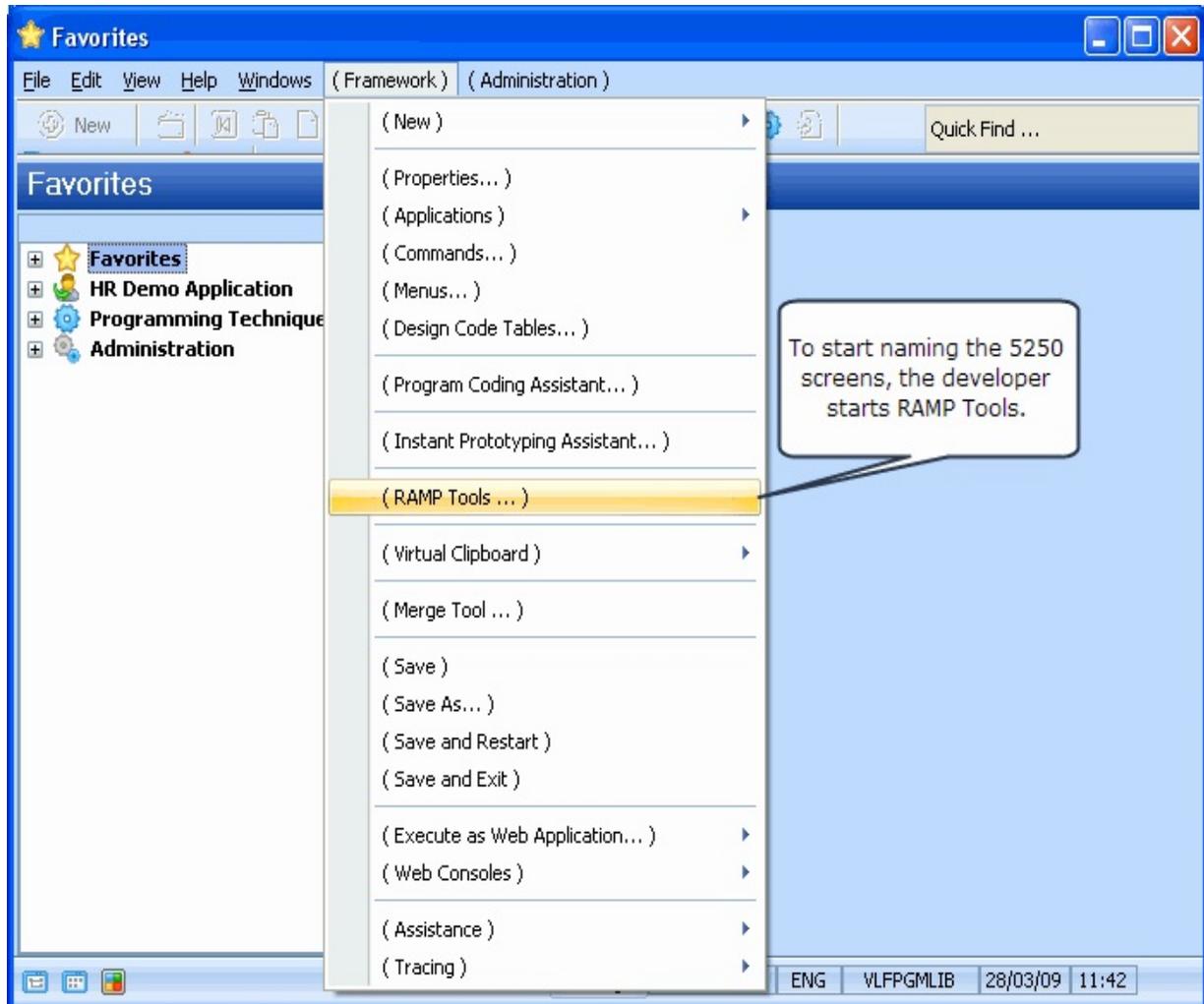
RAMP Stage 2A: Rapidly Modernizing Complete Application Segments

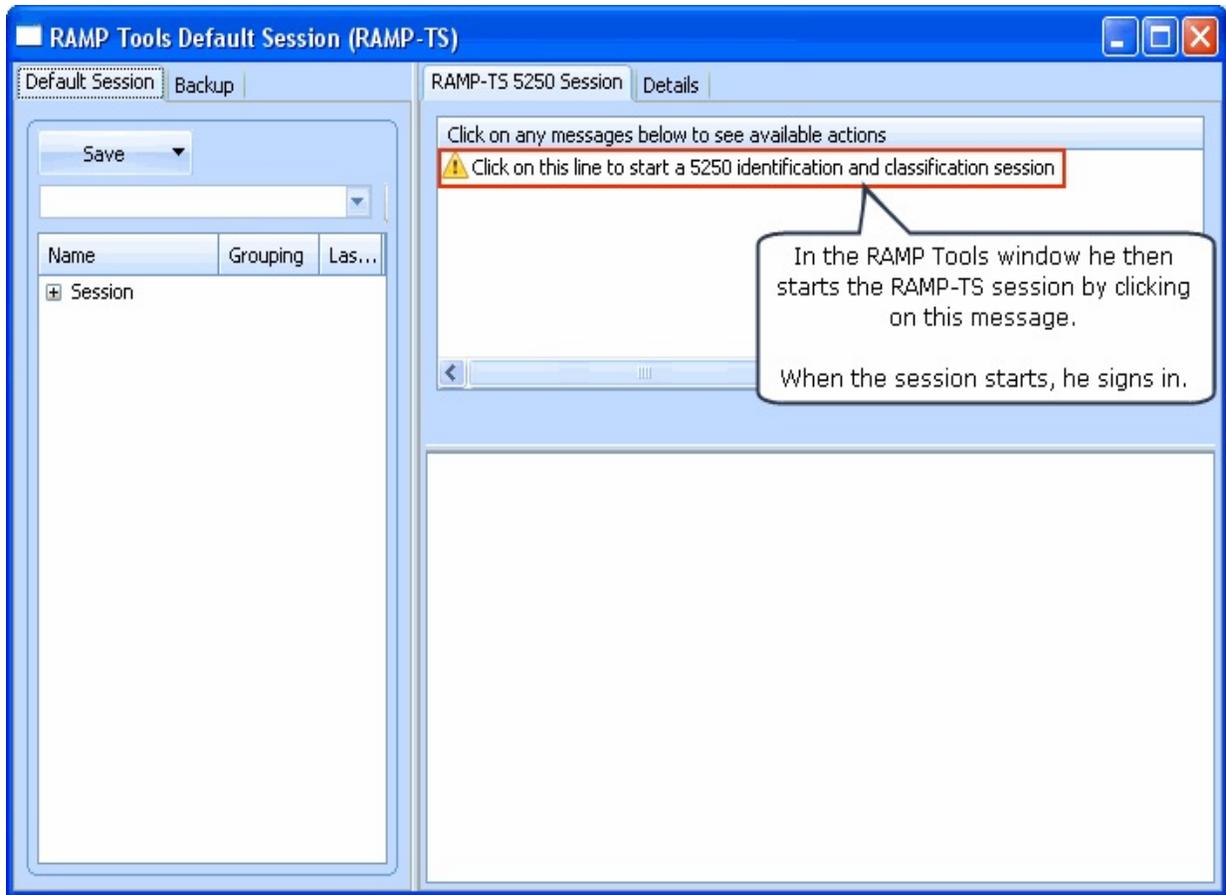
RAMP Stage 2B: Individually Modernizing Frequently Used Screens

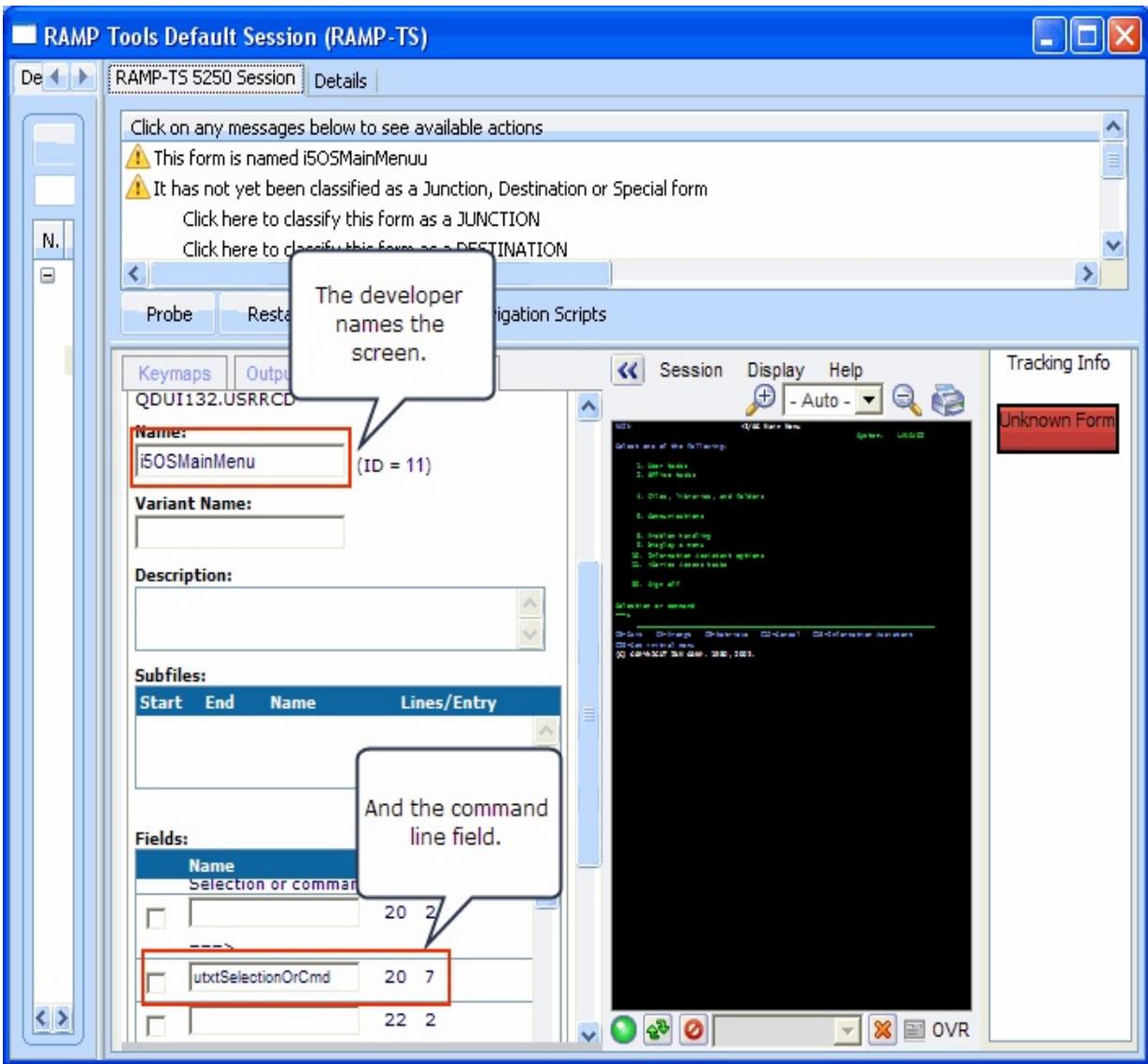
RAMP Stage 2C: Adding Value

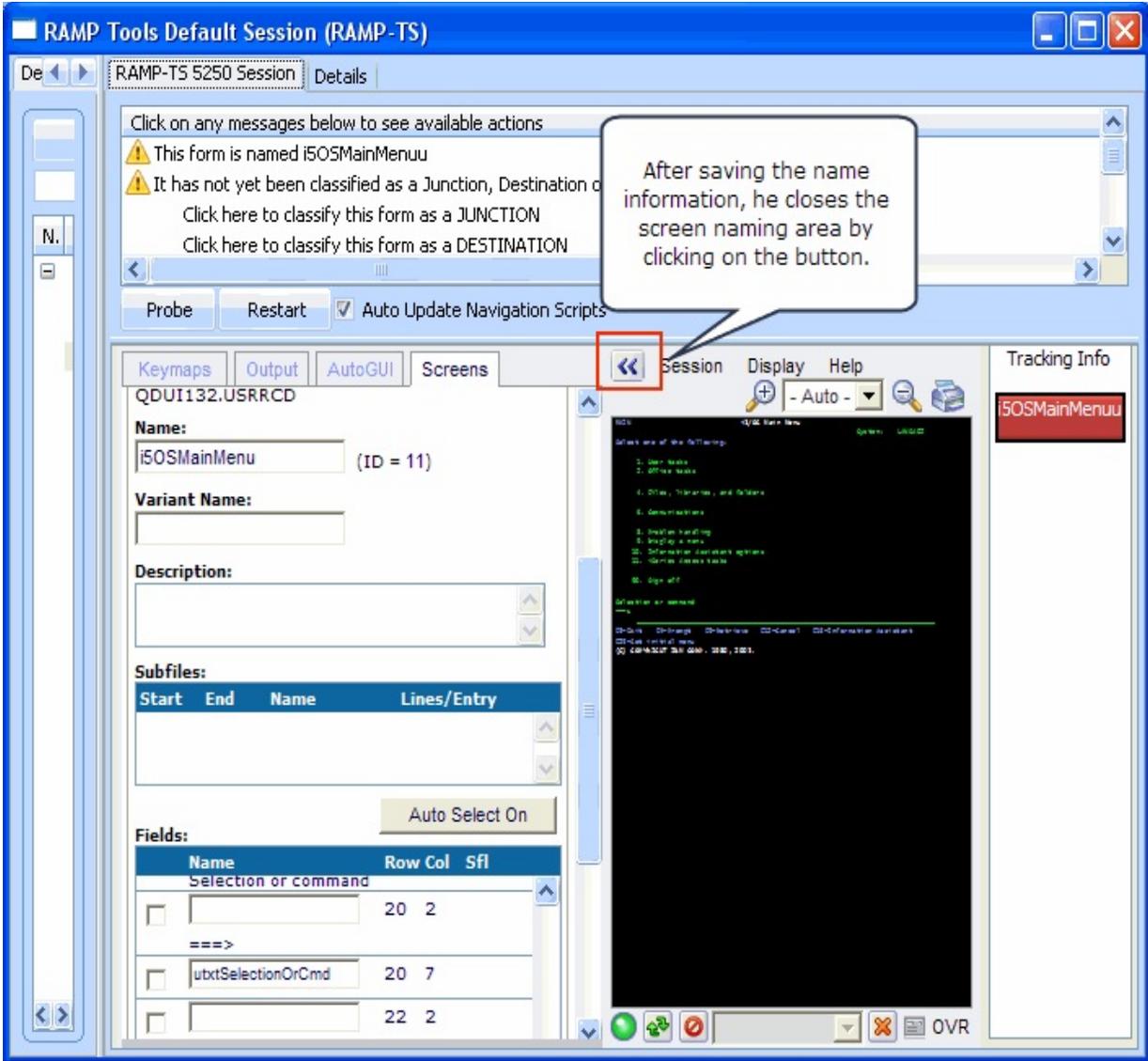
Naming the 5250 Screens

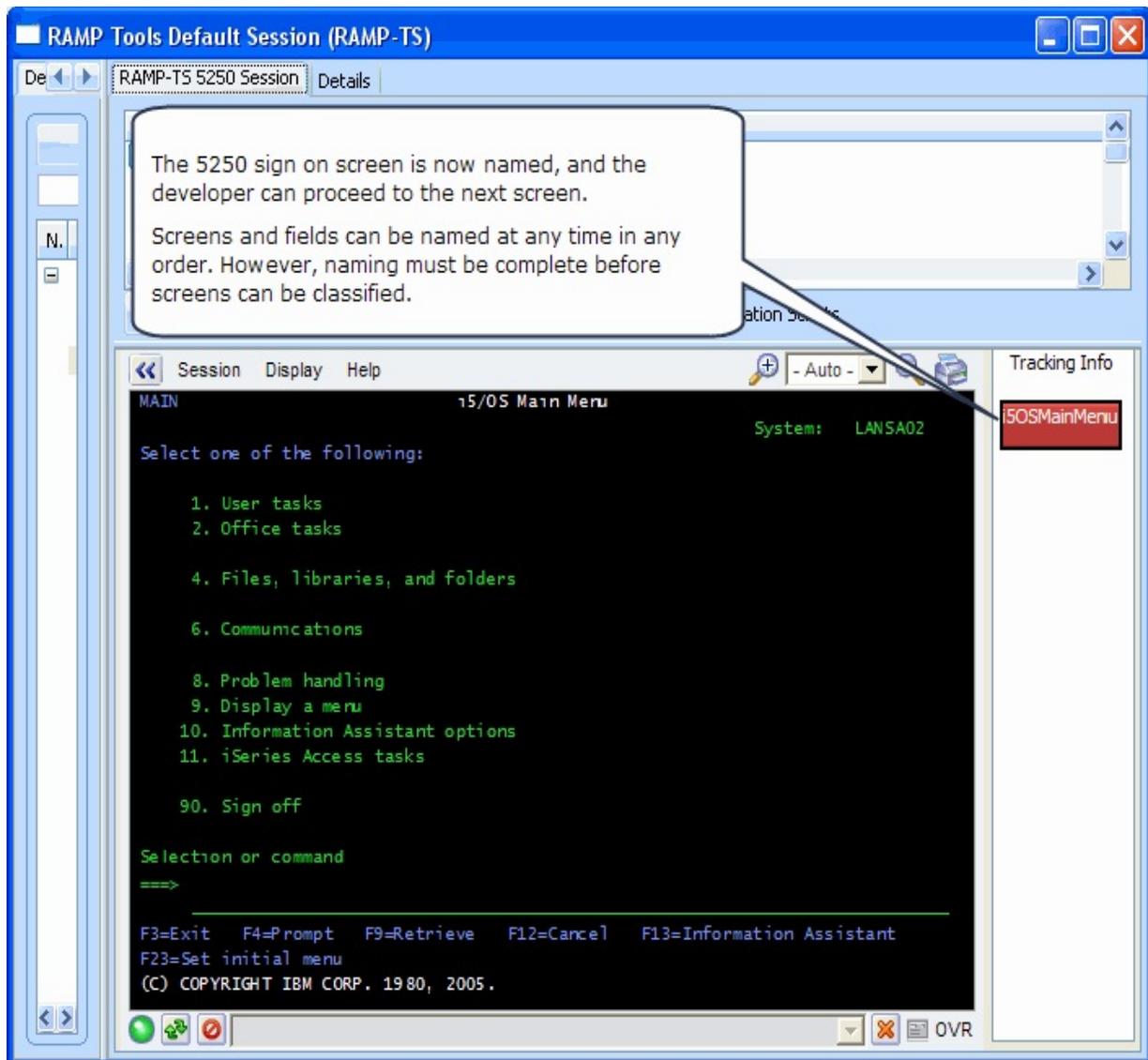
The first step in snapping in 5250 screens into the Framework is to name them:











For more detailed information about naming screens, refer to the tutorials [RAMP-TS002: Rapidly Modernizing Complete Application Segments](#) and [RAMP-TS004: Naming and Classifying the Employee Screens](#).

Tracking and Classifying the Screens

Once the screens and fields to be reused in the modernized application have been named, we can then proceed to classify them and to track the navigation between them:

The screenshot displays the RAMP Tools interface. The main window is titled "RAMP Tools Default Session (RAMP-TS)". The top-left pane shows a tree view with "RAMP-TS 5250 Session" selected. The top-right pane, titled "Details", contains the following text:

Click on any messages below to see available actions

- This form is named MainLogin
- It is defined as a **Junction**
- There are no named items on this form
- The screens that can navigate to this screen are
- The screens that this screen can navigate to are

Below the details pane are buttons for "Probe", "Snapshot (622x334)", and "Restart". A callout box points to the "Details" pane with the following text:

The developer is now going to "demonstrate" to RAMP how to reach the screen that will be snapped into the Framework.

To do this he has to navigate through various 5250 screens, classifying each of them as they are encountered in order to tell the Framework how they will be used within the modernized application.

The 5250 sign on screen is the only exception to this rule, it has been preclassified as a junction screen.

The main display area shows a login screen with the following fields and buttons:

- User:
- Password:
- Reconnect:
- Buttons: Cancel, Login, Advanced

On the right side, there is a "Tracking Info" pane with a button labeled "MainLogin". The bottom status bar shows "aXesTS Signon- Terminal closed" and "OVR".

RAMP Tools Default Session (RAMP-TS)

RAMP-TS 5250 Session Details

Click on any messages below to see available actions

- ⚠ This form is named OSMainMenu
- ⚠ It has not yet been classified as a Junction, Destination or Special

Click here to classify this form as a JUNCTION

Click here to classify this form as a DESTINATION

Click here to classify this form as a SPECIAL

The named items on this form are:
commandline (input)

When the developer reaches the i5/OS Main menu, he classifies it as a junction screen by clicking on the message.

Probe Restart Auto Update Navigation Scripts

Session Display Help

```
MAIN                                i5/OS Main Menu                                System: LANS402
Select one of the following:
1. User tasks
2. Office tasks
4. Files, libraries, and folders
6. Communications
8. Problem handling
9. Display a menu
10. Information Assistant options
11. iSeries Access tasks
90. Sign off
Selection or command
====>
F3=Exit F4=Prompt F9=Retrieve F12=Cancel F13=Information Assistant
F23=Set initial menu
(C) COPYRIGHT IBM CORP. 1980, 2005.
```

Tracking Info

MainLogin

↓ Enter

OSMainMenu

OVR

RAMP Tools Default Session (RAMP-TS)

RAMP-TS 5250 Session Details

Click on any messages below to see available actions

- This form is named OSMainMenu
- It is defined as a Junction
- The named items on this form are:
 - commandline (input)
- The screens that can navigate to this screen are
- The screens that this screen can navigate to are
 - MainLogin - Login Form

Probe Snapshot (622x334) Restart Auto Update Navigation Scripts

Session Display Help - Auto -

```
MAIN                                i5/OS Main Menu                                System: LANS402
Select one of the following:
1. User tasks
2. Office tasks
4. Files, libraries, and folders
6. Communications
8. Problem handling
9. Display a menu
10. Information Assistant options
11. iSeries Access tasks
90. Sign off
Selection or command
-->> |lanssa run ps1tab partition(dem)
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=Information Assistant
F23=Set initial menu
(C) COPYRIGHT IBM CORP. 1980, 2005.
```

Tracking Info

MainLogin

↓ Enter

OSMainMenu

OVR

He then enters a command to move to the next screen.

RAMP Tools Default Session (RAMP-TS)

RAMP-TS 5250 Session Details

Click on any messages below to see available actions

- ⚠ This form is named CodeTableMainMenu
- ⚠ It has not yet been classified as a Junction, Destination or Special form

Click here to classify this form as a JUNCTION

Click here to classify this form as a DESTINATION

Click here to classify this form as a SPECIAL

The named items on this form are:
undefined (output)

Probe Restart Auto Update Navigation Scripts

Session Display Help

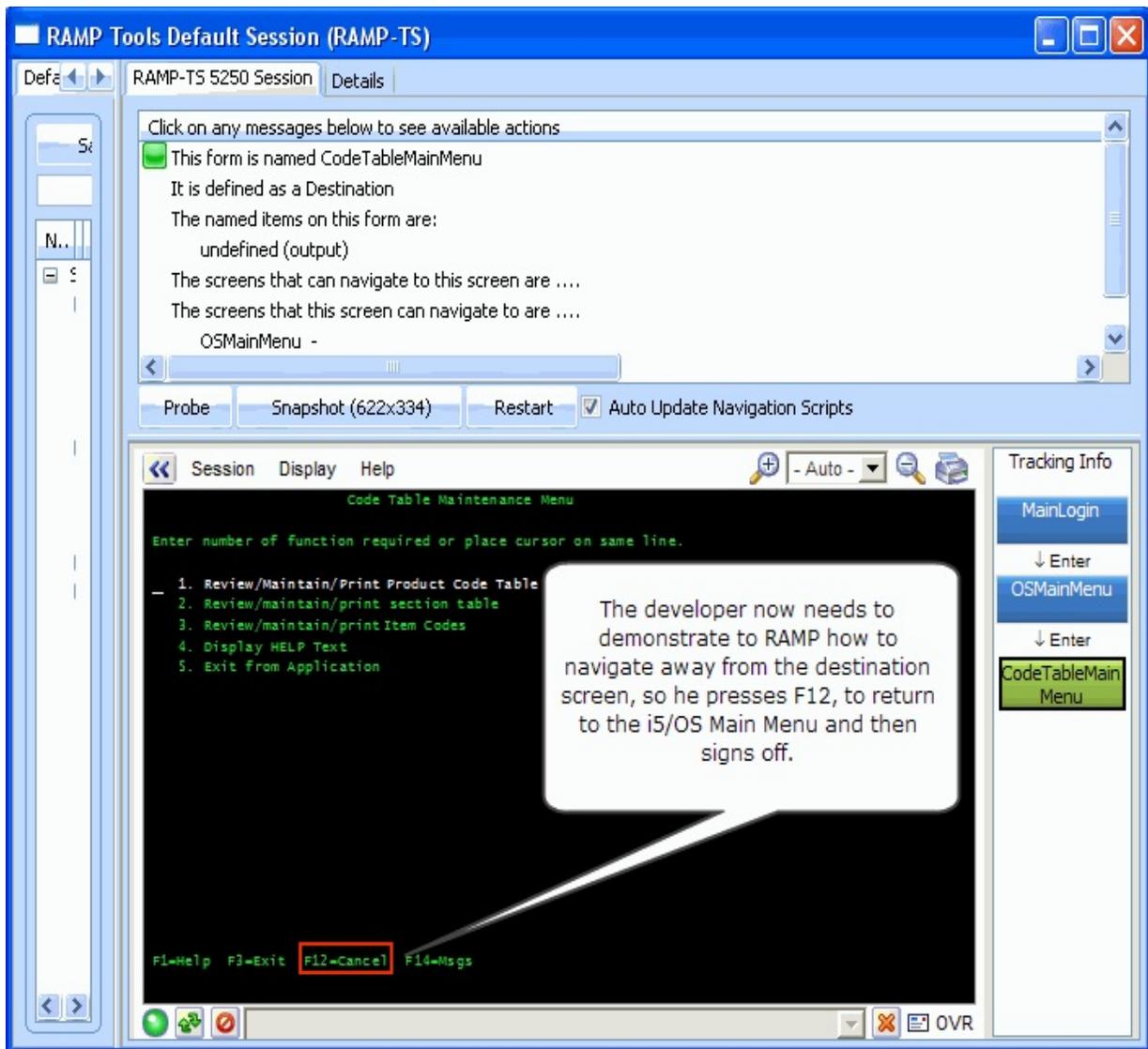
```
Code Table Maintenance Menu
Enter number of function required or place cursor on same line.
1. Review/Maintain/Print Product Code Table
2. Review/maintain/print section table
3. Review/maintain/print Item Codes
4. Display HELP Text
5. Exit from Application

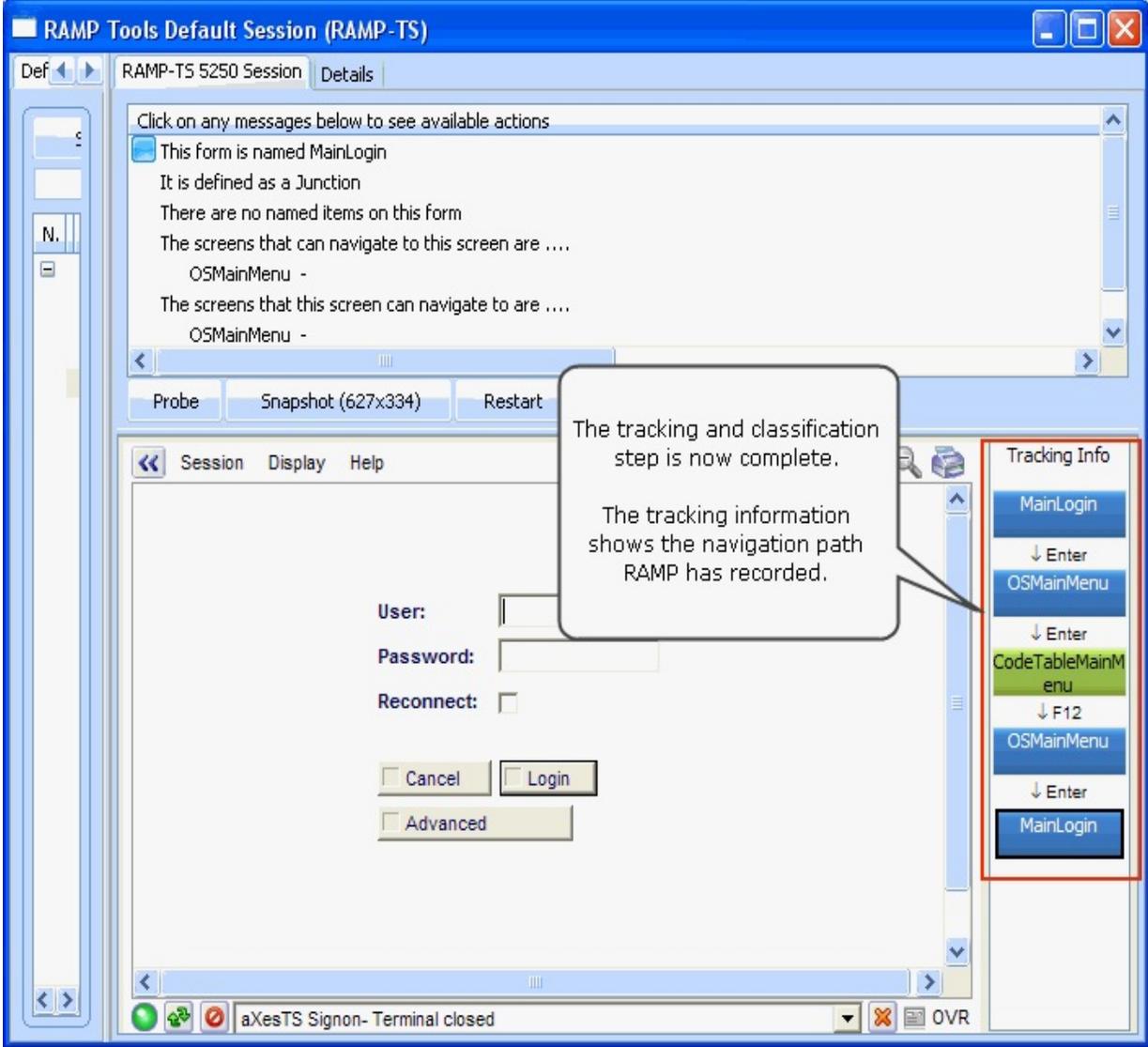
F1=Help F3=Exit F12=Cancel F14=Msgs
```

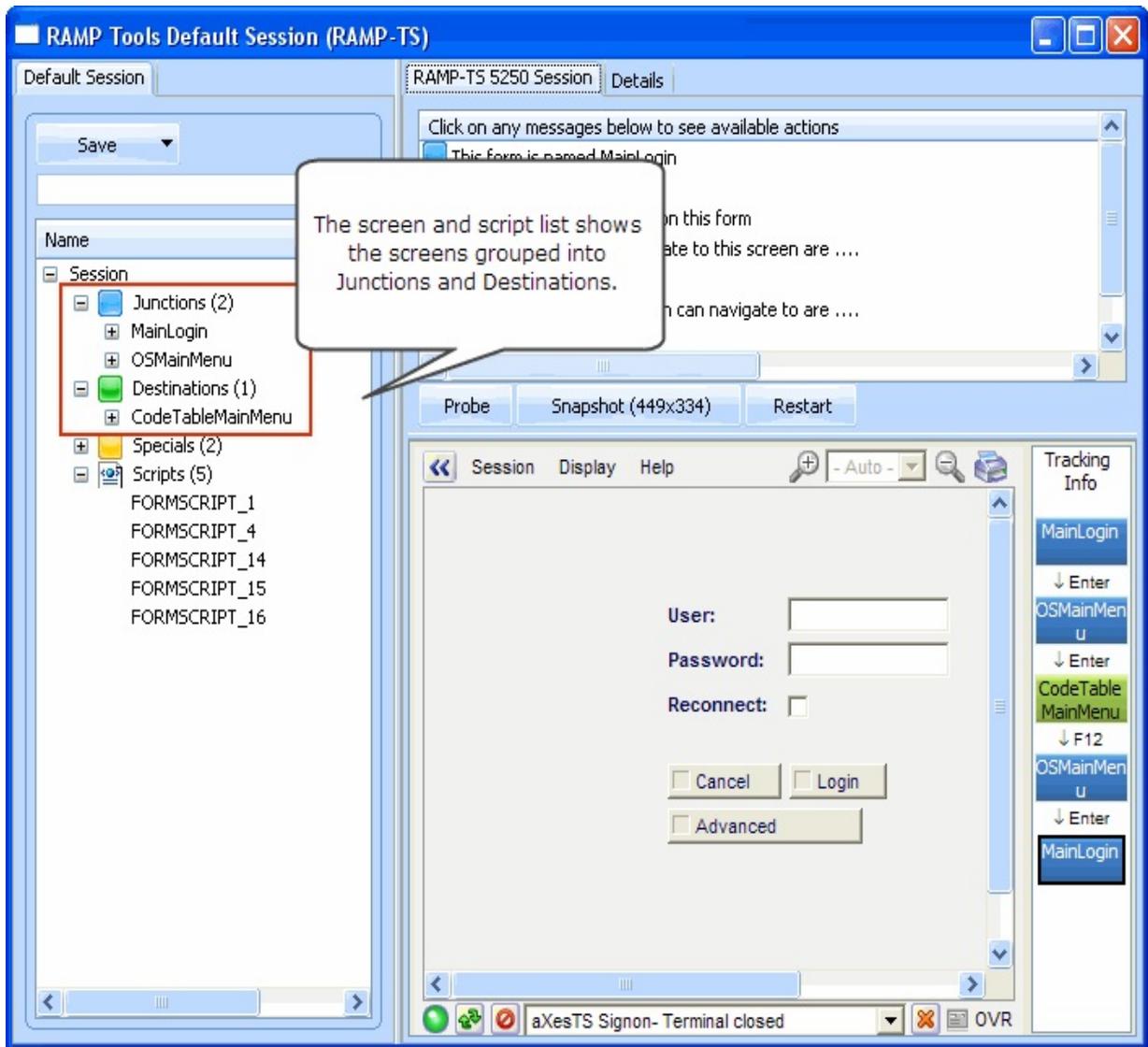
MainLogin
↓ Enter
OSMainMenu
↓ Enter
CodeTableMain Menu

This is the main menu of the Code Tables application segment.

Because this part of the application is seldom used, the developer is not going to modernize the screens in this segment individually. Therefore he classifies the menu screen as a destination screen.







RAMP Tools Default Session (RAMP-TS)

Default Session

Save

Name

- Session
 - Junctions (2)
 - MainLogin
 - Scripts
 - FORMSCRIPT_14
 - OSMainMenu
 - Scripts
 - FORMSCRIPT_15
 - Destinations (1)
 - CodeTableMainMenu
 - Scripts
 - FORMSCRIPT_16
 - Specials (2)
 - Scripts (5)
 - FORMSCRIPT_1
 - FORMSCRIPT_4
 - FORMSCRIPT_14
 - FORMSCRIPT_15
 - FORMSCRIPT_16

RAMP has generated a script for all the screens which control what happens when the screen is displayed and when the user navigates away from the screen.

The scripts are written in Javascript. Javascript was chosen as the scripting language for RAMP because it is powerful, simple and easy to learn.

```
{
  /* =====
  /* ===== vHandle_ARRIVE =====
  /* =====
  /* Handle arrival at this Destination */
  /* oPayload: The payload supplied by the event
  /* oPreviousForm: Reference to previous object

  vHandle_ARRIVE: function(oPayload, oPreviousForm
  {
    var bReturn = true;

    SHOW_CURRENT_FORM(true); /* show the form in
    HIDE_5250_BUTTONS(); /* Hide any 5250 sty
    GET_FORM_MESSAGE(22); /* Extract messages
    SETBUSY(false); /* Last thing done -

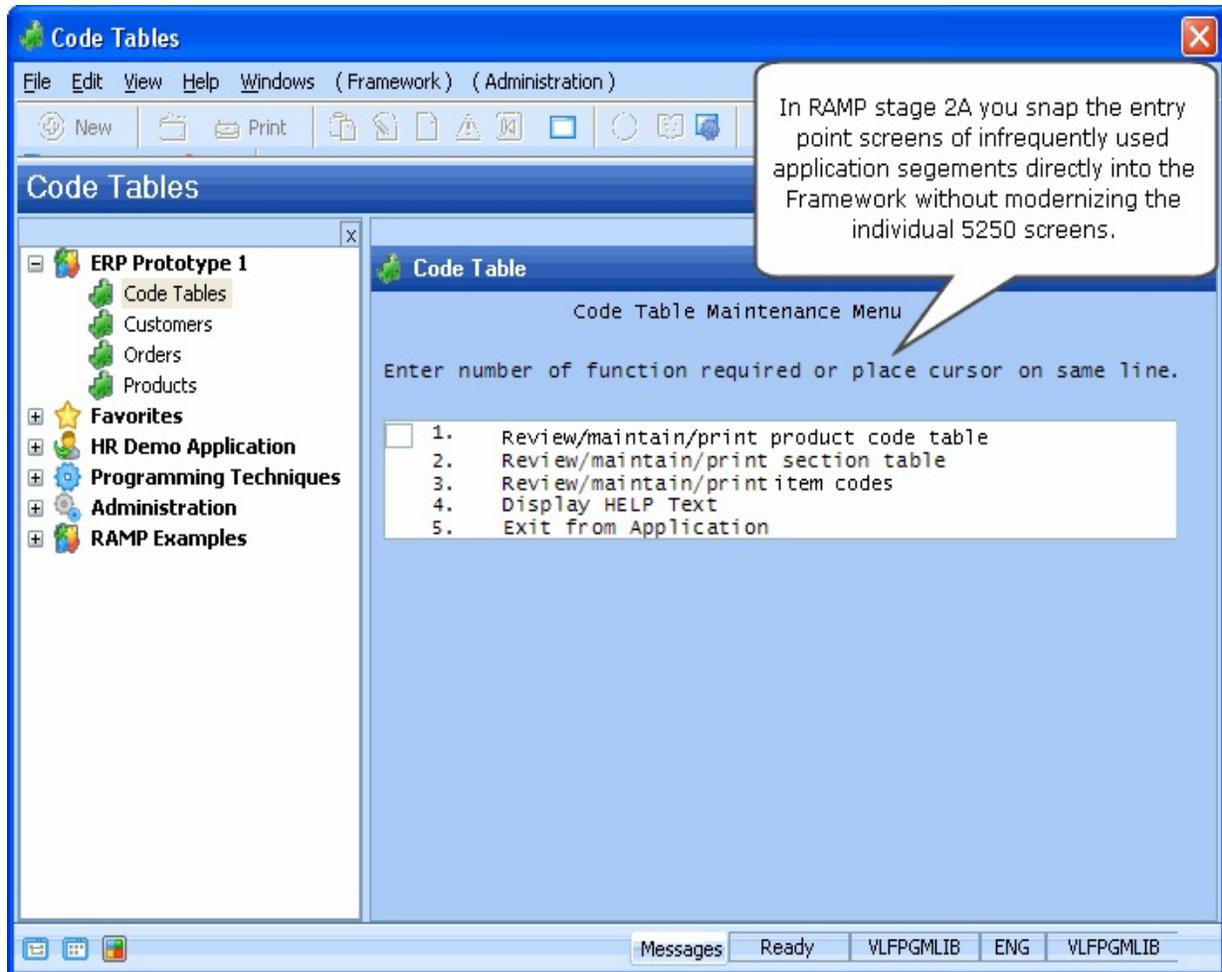
    /* <ARRIVE /> - Do not remove or alter this ]

    return(bReturn);
  },

  /* =====
  /* ===== BUTTONCLICK =====
  /* =====
  /* sButton: The button that was clicked */
```

RAMP Stage 2A: Rapidly Modernizing Complete Application Segments

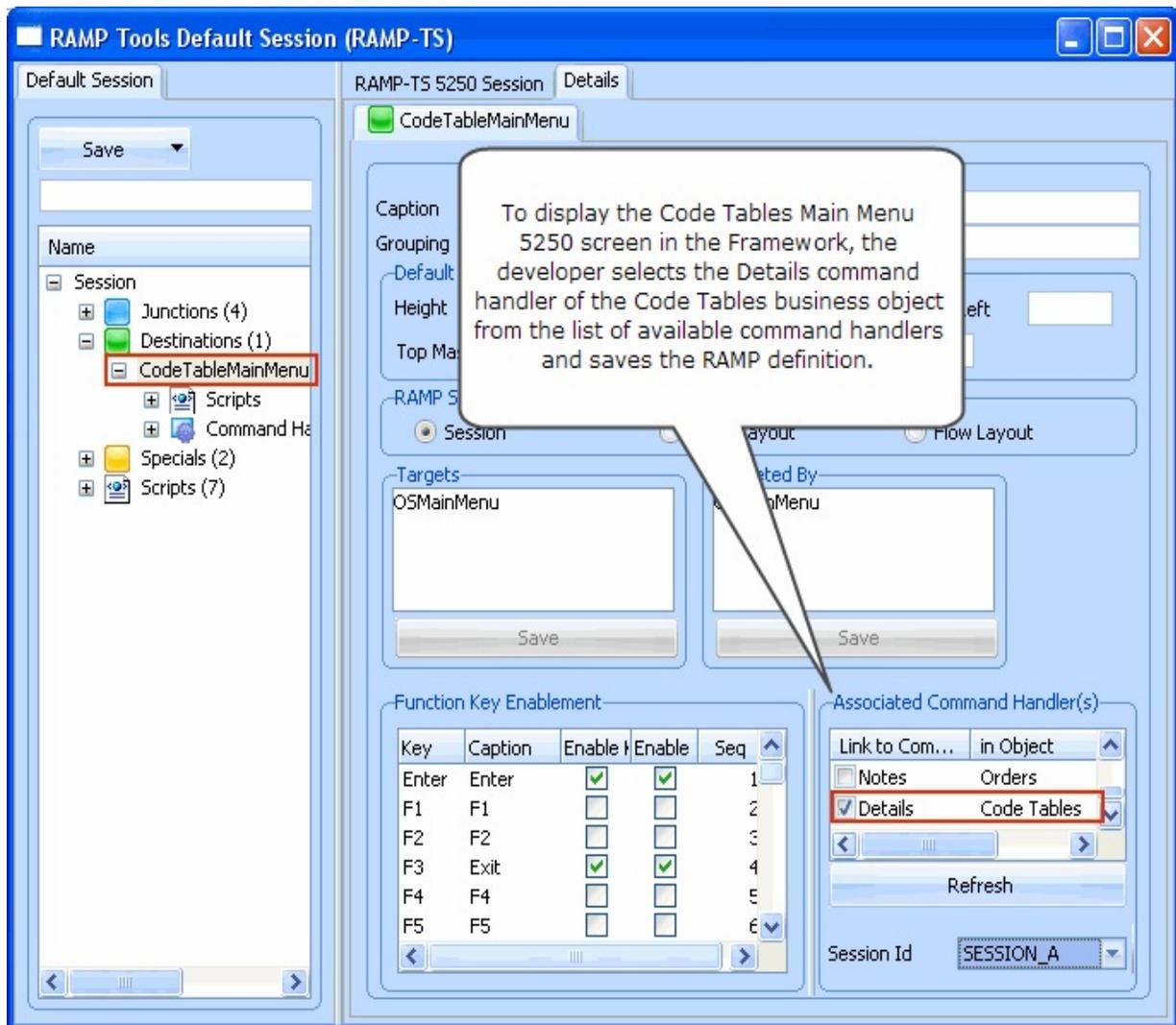
During the prototyping stage you will have identified those segments of the application which are used infrequently, and you have named and choreographed the navigation to the entry point screens of those segments. Now you snap them in the Framework.



How to Do It?

How to Do It?

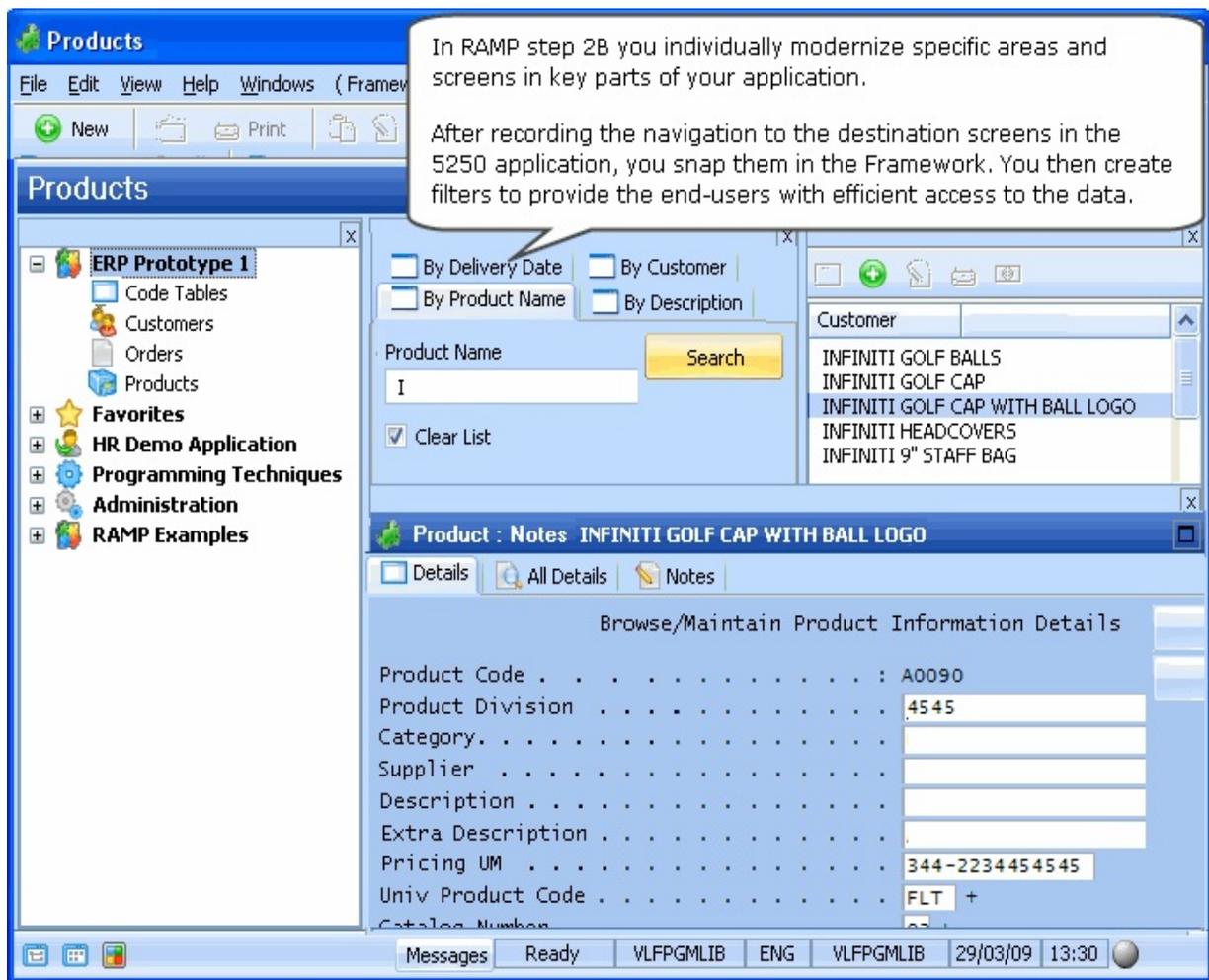
After the 5250 entry point screens have been named and the navigation to and from them has been recorded, they are simply snapped in the Framework:



See tutorial [RAMP-TS002: Rapidly Modernizing Complete Application Segments](#).

RAMP Stage 2B: Individually Modernizing Frequently Used Screens

During the prototyping stage you will have identified those segments of the application which are used often, and you have named and choreographed the navigation to the destination screens of those segments. Now you snap them in the Framework, and create filters to provide flexible and efficient access to the data.



How to Do It?

How to Do It?

See tutorials:

[RAMP-TS003: Creating a Data Filter for Employees](#)

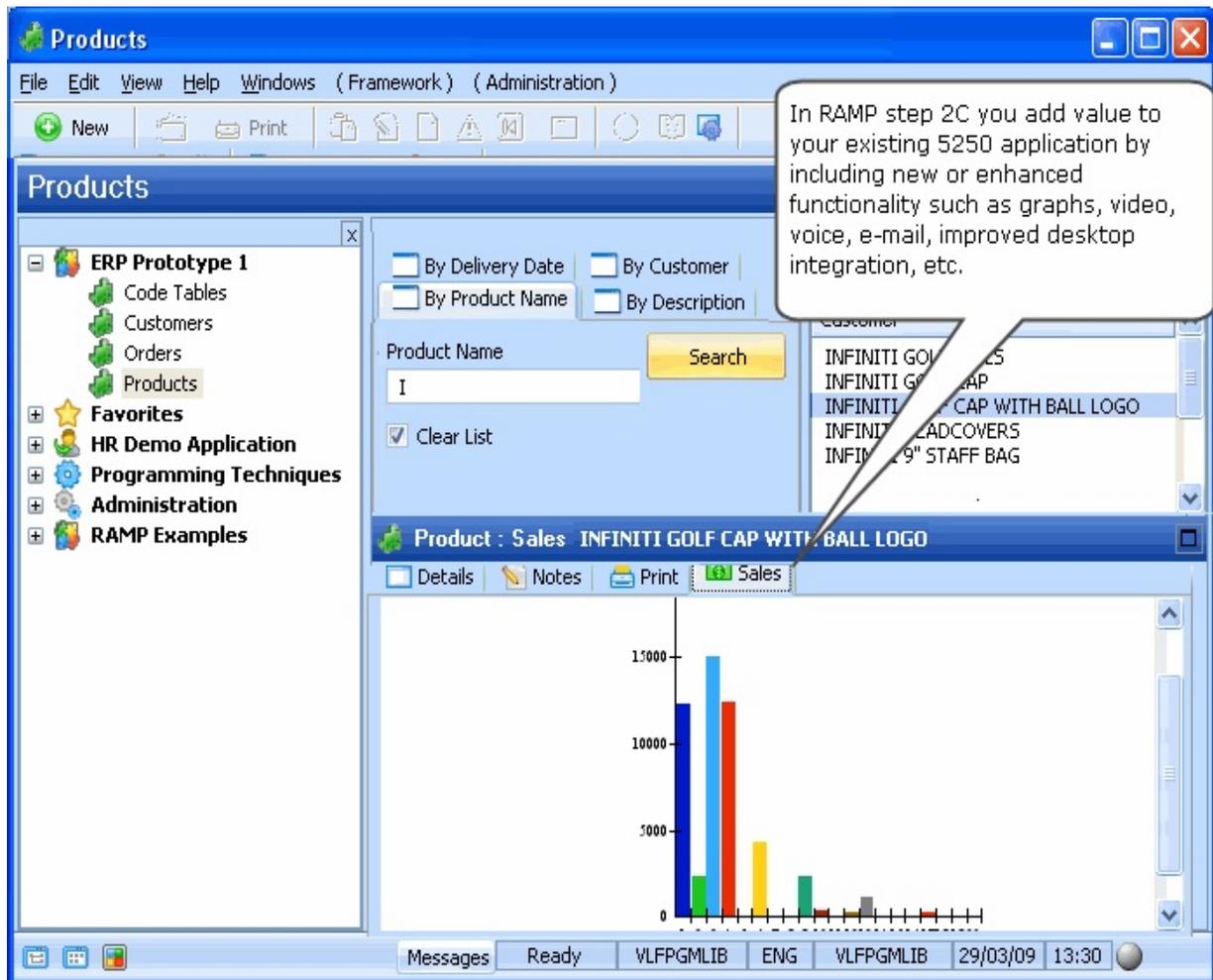
[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

RAMP Stage 2C: Adding Value

In this step you selectively enrich your existing 5250 application by adding new or enhanced functionality such as video, voice, e-mail, improved desktop integration etc.



Adding new Visual LANSA components opens up a whole new range of ways that modern IT technologies can be applied to improving business processes.

What Adds Value?

How to Do It?

What Adds Value?

If you do not add real business value to your modernized application, your project may not succeed.

Things that make or save money for the business add value. They range from business process and productivity improvements, lower training costs to happier staff. Similarly, things that allow end-users to do their jobs faster, better and smarter, with a higher level of personal satisfaction add value.

Introducing trivial value items such as using a tree to replace 5250 menu navigation with a few radio buttons and drop downs may be nice to have, but to an end-user they might even have a negative value.

Some of the ways to add real value to an existing 5250 application are:

Using smart and powerful filters to access data Filters add value because they can easily do things that the existing 5250 application cannot do, and because they can be tailored to exactly match common end-user business processes.

For example, if you take a basic product enquiry 5250 screen and add a powerful filter over the product master and order details files, you can add a lot of value to an end-users working life, for example by making it possible to find all products on order today, or on back order, search by name, search by category, etc.

Filters that allow end users to define their own repetitive custom searches (for example every Monday I want to see all products sold last week) or dynamically create commonly used lists (for example all products marked as hazardous) also add significant real value.

If your filters simply mimic the 5250 application you will not be adding much value.

Consolidating information Often 5250 applications require people working in warehouses and offices to extract information from multiple sources and consolidate it into management reports or use it for input to disparate

and non-integrated applications.

If you can consolidate these tasks on a single form with a single click or two, you will add real value to management and to users.

Integrating with desktop applications

Add value by integrating functionality such as e-mail that aids communications between users and customers, and MS-Excel that aids in reporting and analysis tasks.

Reducing repetition and rekeying

Lots of 5250 applications still require users to rekey information because the underlying 5250 applications are not integrated.

If you can reduce or eliminate this you will add value to management and users. Smart prompting and pre-filling 5250 screens, sometimes from a user-definable list of templates, are also good ways to add value.

The things listed above are simple examples of what is at the root of most effective business process re-engineering. It is about what you already have and then reusing it in a new improved way.

You need to consider all this at [Stage 1: Creating a Modernization Framework](#).

How to Do It?

See tutorials:

[RAMP-TS010: Using Special Field Handlers](#)

[RAMP-TS011: Snapping in Shipped Documents Command Handler](#)

[RAMP-TS012: Snapping in Shipped Notes Command Handler](#)

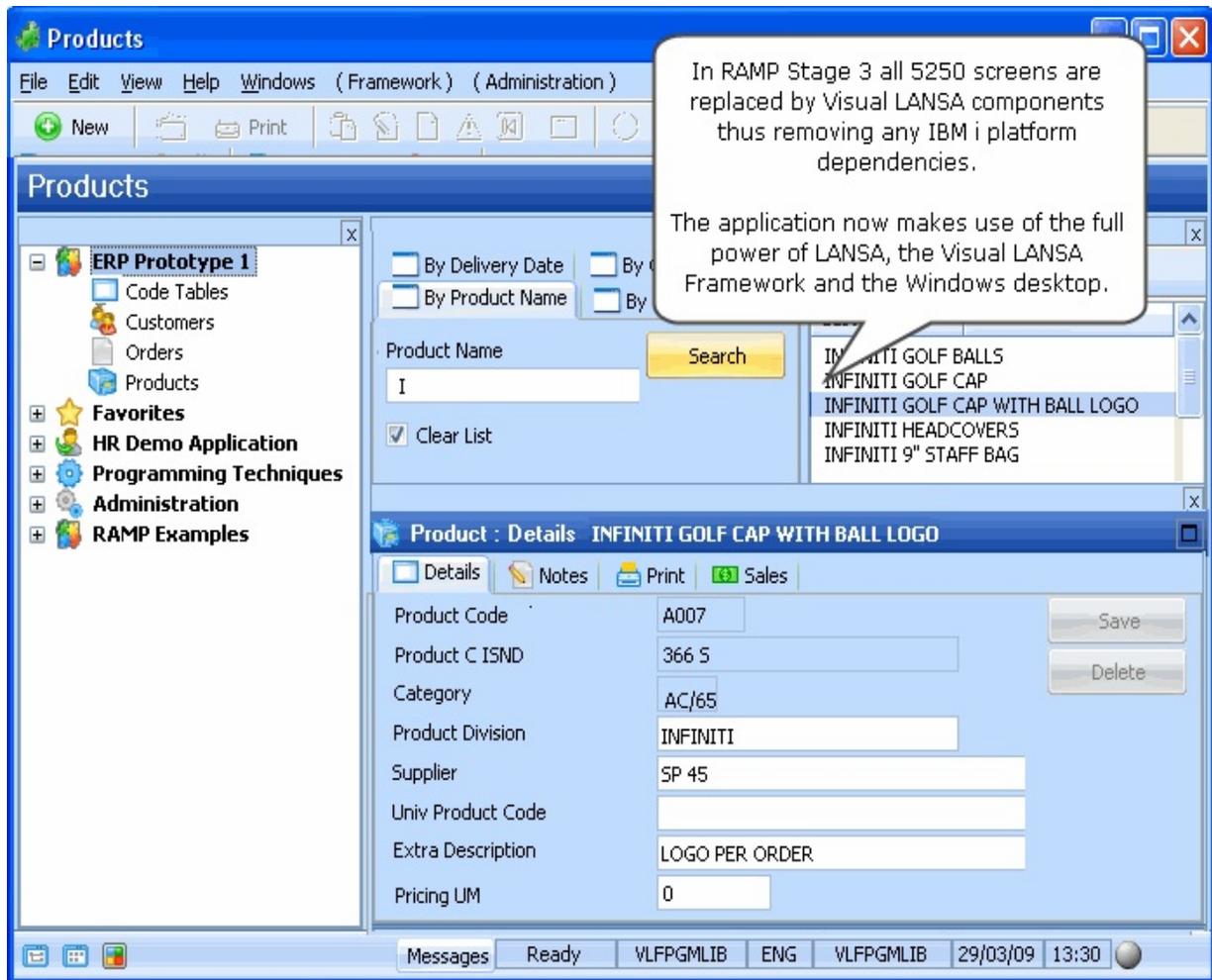
[RAMP-TS014: Snapping RAMP Screens into the HR Demo Application](#)

[RAMP-TSAD01: Using Buttons to Add Value to an Application](#)

[RAMP-TSAD03: Special Field Handling](#)

Stage 3: Removing IBM i Platform Dependencies

RAMP Stage 3 is an optional stage if you do not need to support server platforms other than the IBM i. The final objective of this stage is to replace all the 5250 destination screens with platform independent Visual LANSA components, in other words it involves re-engineering your existing 5250 application.



This brings some significant benefits:

The first is that their entire application is no longer dependent upon RPG, COBOL, DDS, or for that matter executing using an IBM i or i5 server. This may be of critical importance if you need to sell your application to customers who want to use Windows or Linux servers.

The re-engineering exercise may be undertaken using any technology or

approach that LANSA supports. (eg: SOA,MVC).

Stage 3 supports a rapid multiple incremental delivery approach that lowers time to market and risk.

Stage 3 opens up the full power of LANSA, the Visual LANSA Framework, the Windows desktop and the web browser to your applications.

Stage 3 is what makes RAMP a long term strategic approach to application modernization. It provides a clearly defined path forward into the future rather than a tactical “step-sideways” solution.

Prerequisite Skills

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

[Framework tutorials](#)

[Visual LANSAs tutorials](#) - you will need to know how to create, compile and check programs into your System i server using the Visual LANSAs 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).

Warnings and Disclaimers

The Clever versus Risk Dilemma

As you progress with RAMP-TS you will probably notice that the VLF and the RAMP-TS server use internal processing models that are exposed in JavaScript, the HTML DOM, etc. You can do this by looking at the shipped code or by using all sorts of tools explore the models.

This will lead you to the "Clever versus Risk" dilemma, which is this

You can probably write clever logic to traverse, modify, reuse or alter the internal processing model of the framework or RAMP-TS. You may have perfectly valid reasons for doing this and you may be able to produce some very useful and powerful results.

However, in doing this you are taking a very clear risk. The risk is that in a future version of the framework or RAMP-TS the internal models may change in behaviour or content. These changes may render your use of the internal model useless, malfunctioning or even damaging.

From the product support and maintenance perspective LANSA will support published interfaces (ie: things documented in this, or the VLF guide).

If you traverse, modify, reuse or alter the internal processing model of the framework or RAMP-TS you need to be aware that you are taking a risk, the cost of which must be borne by you or the organization you work for.

There is no issue with your right to take such actions, but you need to be very clear about the risk involved and about who will pay for any subsequent consequences.

What's New

This version of RAMP supports LANSAs Version 13.

To review new features in previous RAMP versions, see:

[New Features in EPC 868 Version of RAMP](#)

[New Features in EPC 843 Version of RAMP](#)

New Features in EPC 868 Version of RAMP

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

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.

New Functions

[LOCK_FRAMEWORK](#) Function

[UNLOCK_FRAMEWORK](#) Function

[RESTART_LAST_NAVIGATION](#) Function

To review new features in previous RAMP versions, see:

[New Features in EPC 843 Version of RAMP](#)

New Features in EPC 843 Version of RAMP

This section outlines new features in EPC843 version of RAMP-TS:

SSL Support for RAMP-TS

SSL (Secure Socket Layer) support can now be added to RAMP-TS. Check the See the [Use HTTPS](#) option in the Framework server options.

Note: To use SSL with RAMP-TS from VLF-WEB or VLF-NET applications, the VLF Web server also has to operate under SSL.

Stop Values Saved to the Virtual Clipboard Persisting Beyond the Current Execution

A new parameter for the [AVSAVEVALUE Function](#) has been added so that, when set to FALSE, values saved to the VLF Virtual Clipboard can be stopped from persisting to further invocations of a RAMP-TS application. The default value is TRUE.

Control when to recognize subfiles as tables

The new [OVRSF LAUTOGUI Function](#) allows you to turn the

Switch From RAMP-TS Command Handlers to Other Business Objects

The new [AVSWITCH Function](#) enables a switch from the current screen to another business object and executes a nominated command once there. This is a basic implementation of the VLF Switch method.

Private Version of the SHARED Object

A private version of the SHARED Object can now be used when RAMP-TS screens are kept in a Private Definition Folder. Check the [Contains SHARED Object](#) option in the RAMP-TS (Terminal Server) group box to indicate to RAMP-TS that the SHARED Object file is in the Private Definitions Folder. See [Contains SHARED Object](#).

Note: If a Private Definition Folder is not nominated then the SHARED Object file is loaded from its default location.

Axes system flag Recognise
subfiles as tables on and off on a
screen-by-screen basis.

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 LANSAs applications	Can you create, compile and execute Visual LANSAs (VL) applications and check them into your System i server?
RAMP choreographer license	To choreograph RAMP screens	Use LANSAs REQUEST(LICENSE) on your System i server and look for a valid "AXX" or "Ann" license.
Axes-TS license	To display 5250 sessions via the Axes-TS server	Use the IBM i command WRKLICINF PRDID('1AXES*'). Note the asterisk. Depending on what features have been installed, the resulting screen may will show these features: Base, Terminal Server, Application Server, Spooler File Server, Intersession, Terminal Server etc.
LANSAs Super Server license	To access remote data and programs on your System i via non-5250 interfaces.	Use LANSAs REQUEST(LICENSE) on your System i server and look for a valid "LXX" or "Xnn" license.

To obtain any type of license contact your LANSAs product vendor.

Installation and Configuration

Installation on the Server

Backup Strategy

Configuration

When Many Developers Work on the Same Application

Installation on the Server

See the Framework Guide for detailed instructions for installing the Framework.

You need to:

First [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 LANSAs for the Web.

Next [Install and Configure the Framework on Visual LANSAs Workstations](#).

Then [Install and Configure RAMP-TS / aXes-TS on your IBM i Server](#).

Lastly [Verify your RAMP-TS/aXes-TS Installation](#)

[Backup Strategy](#).

If you have used RAMP-NL (newlook), you may be interested in the frequently asked question [How is a Framework associated with RAMP-TS or RAMP-NL?](#)

Install and Configure RAMP-TS / aXes-TS on your IBM i Server

Before you can use RAMP-TS you need to install and configure aXes terminal server on your IBM i server.

Before Installation

Before starting the installation, please read the planning section in the aXesQuick Start guide.

1. If this product is already installed, refer to aXes Quick Start guide for how to upgrade, or contact your vendor. (To check if it aXes is installed, run the command GO LICPGM, display installed license programs and look for product – 1AXES01).
2. If aXes it not installed, download it from www.axeslive.com and click on download.
3. Install aXes following the instructions in the aXesQuick Start guide.

Verify your RAMP-TS/aXes-TS Installation

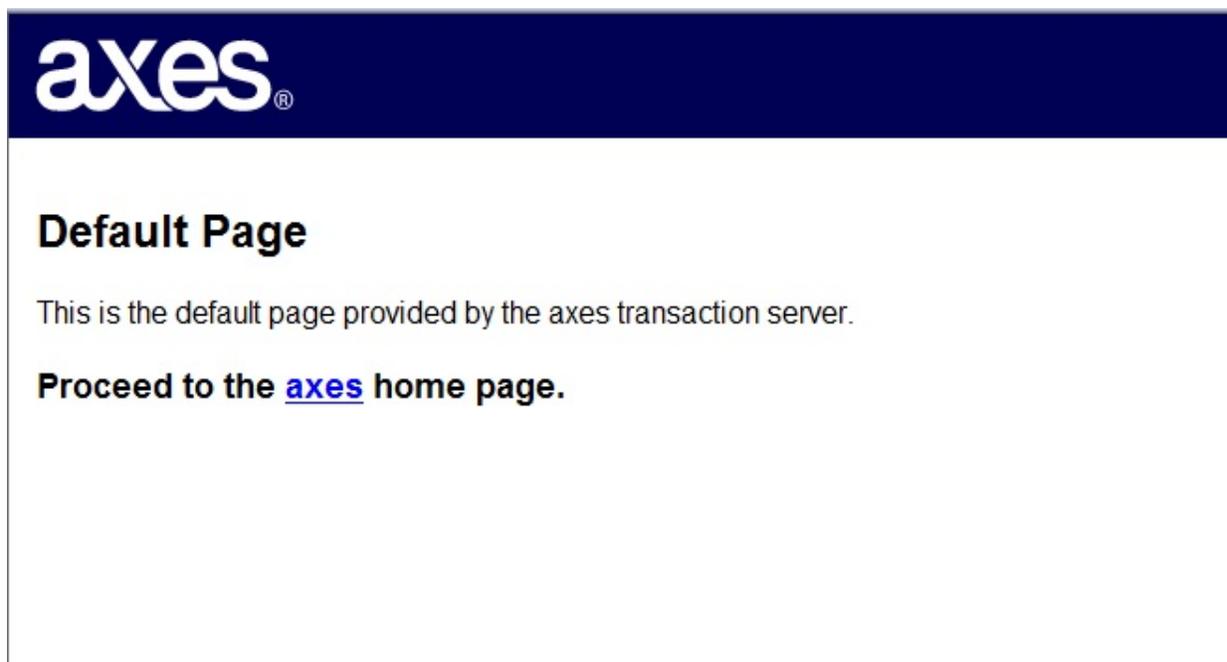
After installing RAMP-TS/aXes on your IBM-I server you should verify the installation.

Please do not proceed any further with using RAMP-TS until you can successfully complete the following verification steps:

1. Open a browser window and enter a URL like this, specifying the host and port of your aXes-WS server:

```
http://<host>:<port number>
```

The resulting screen should look like this:



2. Click on the aXes home page link.

The result should look like this:

axes[®]

Home | Application Server | Terminal Server | Terminal Sessions

Server administration

This application provides valuable information to your iSeries server administrators such as:

- view real time server statistics
- close one or multiple user sessions

By default, the username/password combination for the menu options above is admin/admin, and dev/dev for the TS development session link. It is recommended that you change these values using the instructions in the [axes user guide](#) in section 2, "Managing axes security".

To view and manage user sessions, click on the "TS administration" link above, then click on "TS sessions". Click on a users name to view their session. To close all sessions, click the "All" text above the list of checkboxes, then press the close button at the bottom of the screen.

Documentation

Follow the links on the right to access valuable axes technical documents.

launch axes sessions

- :: [TS terminal session](#)
- :: [TS development session](#)
- :: [Spool file session](#)
- :: [Data explorer session](#)

launch axes tests

- :: [Test application server](#)

axes documentation

- :: [User guide](#)
- :: [Quickstart guide](#)
- :: [Reference manual](#)

3. Click on the TS terminal session option on the right hand side, in the launch aXes session's group. A separate window should appear that allows you to sign on to your IBM i server using your normal user profile and password:

axes[®] TS terminal session

User:

Password:

Reconnect:

Cancel Login

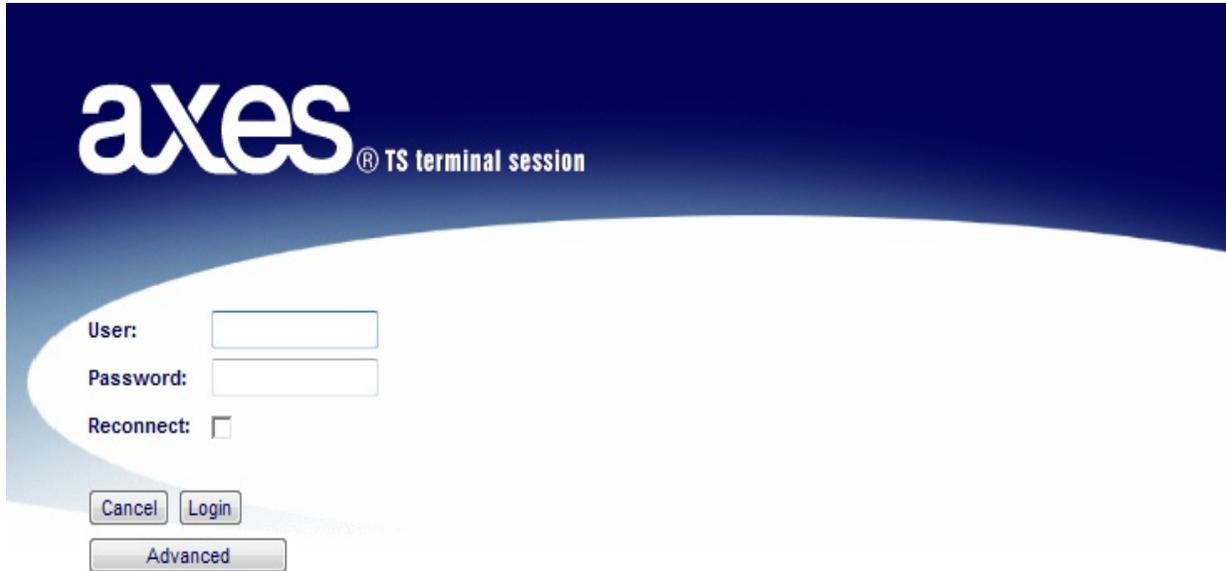
Advanced

4. After verifying you can sign on and access your IBM i server signoff and close the window.
5. Now click on Click on the TS development session option on the right hand

side, in the launch aXes session's group.

6. When the web logon dialog appears sign on as user dev with password dev.

The resulting screen should look like this.



7. Verify that you can log on to your IBM i server, using your user profile and password.

Backup Strategy

The library AXES and the IFS folder aXes and all its subfolders must be included into your daily backup cycle. Failure to do this could lead to a significant or total loss of developer work.

See [Where and how are my RAMP-TS screen identifications kept?](#)

Configuration

[Verify Browser Security Settings](#)

[Configure RAMP](#)

Verify Browser Security Settings

To use RAMP-TS it is necessary to bypass browser cross-domain security (security relating to documents accessing documents from a different domain).

In Internet Explorer cross-domain security is bypassed by specifying the same Host name in the VLF and Axes URL. For example, let's say hostabc resolves to 10.10.2.181.

This URL is OK because both refer to hostabc regardless of reverse proxy configuration and hostabc added to the trusted sites:

http://hostabc:81/images/privatefolder/EX1_Test_ENG_BASE.HTM?Developer=Y+TSIPAddress=hostabc+TSPortNumber=8080

This URL is also OK because both refer to 10.10.2.181 regardless of reverse proxy configuration and 10.10.2.181 added to the trusted sites:

http://10.10.2.181:81/images/privatefolder/EX1_Test_ENG_BASE.HTM?Developer=Y+TSIPAddress=10.10.2.181+TSPortNumber=8080

These URLs are NOT OK:

http://10.10.2.181:81/images/privatefolder/EX1_Test_ENG_BASE.HTM?Developer=Y+TSIPAddress=hostabc+TSPortNumber=8080

http://hostabc:81/images/privatefolder/EX1_Test_ENG_BASE.HTM?Developer=Y+TSIPAddress=10.10.2.181+TSPortNumber=8080

For other browsers this is done by setting up a reverse proxy:

[Set Up Reverse Proxy for Chrome, Safari and Firefox](#)

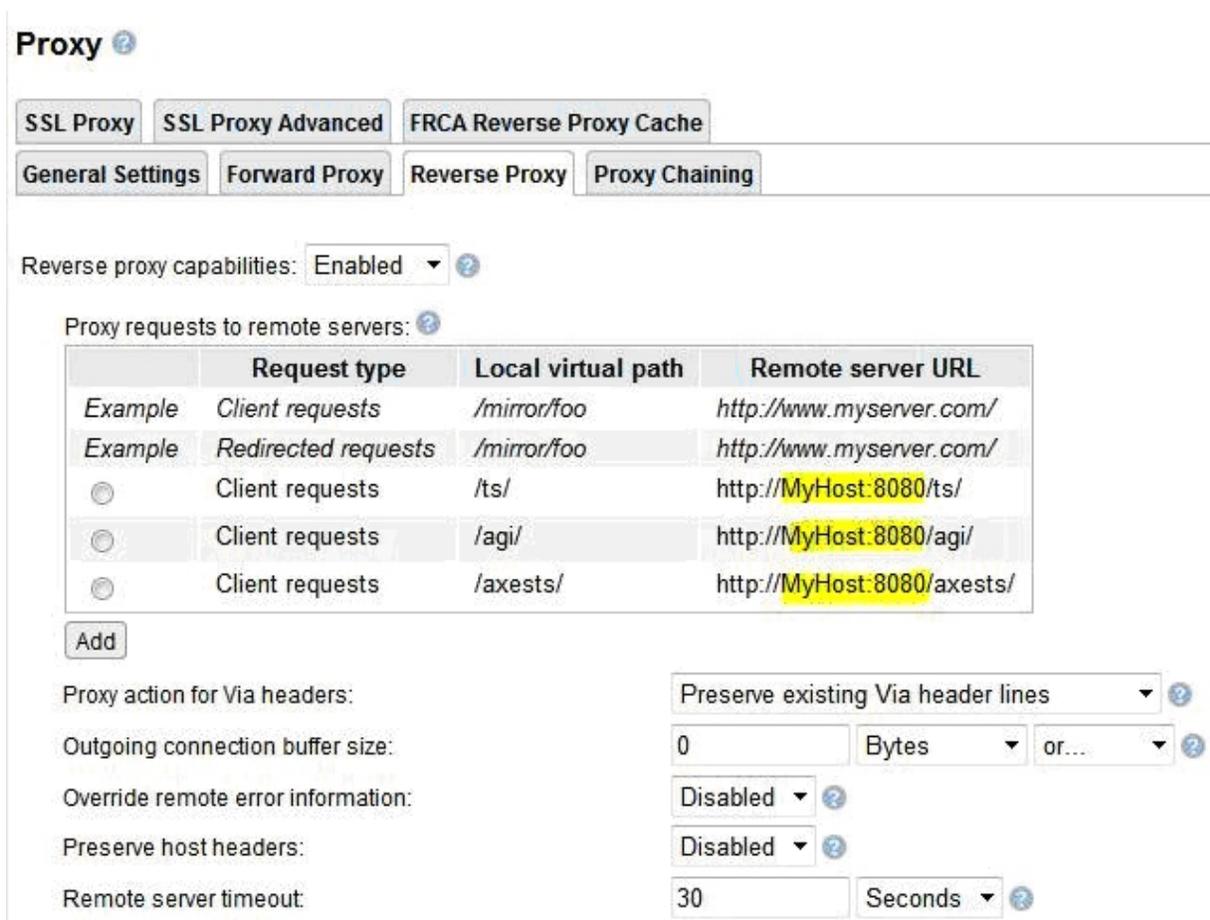
Set Up Reverse Proxy for Chrome, Safari and Firefox

Perform this step on every PC from which RAMP-TS used.

Domain refers to the Host:Port combination. For example if the VLF uses a host MyHost in port 81, the VLF domain is MyHost:81 and if RAMP-TS (aXes) also uses MyHost but in port 8080, the RAMP-TS domain is MyHost:8080.

Therefore the VLF and RAMP-TS are accessing different domains.

Therefore it is necessary to use the web server Reverse Proxy feature to bypass cross-domain security. The Reverse Proxy settings for the sample host names look like this in the IBM i Admin instance:



Proxy ?

SSL Proxy SSL Proxy Advanced FRCA Reverse Proxy Cache

General Settings Forward Proxy Reverse Proxy Proxy Chaining

Reverse proxy capabilities: Enabled ?

Proxy requests to remote servers: ?

	Request type	Local virtual path	Remote server URL
Example	Client requests	/mirror/foo	http://www.myserver.com/
Example	Redirected requests	/mirror/foo	http://www.myserver.com/
<input type="radio"/>	Client requests	/ts/	http://MyHost:8080/ts/
<input type="radio"/>	Client requests	/agi/	http://MyHost:8080/agi/
<input type="radio"/>	Client requests	/axests/	http://MyHost:8080/axests/

Add

Proxy action for Via headers: Preserve existing Via header lines ?

Outgoing connection buffer size: 0 Bytes or... ?

Override remote error information: Disabled ?

Preserve host headers: Disabled ?

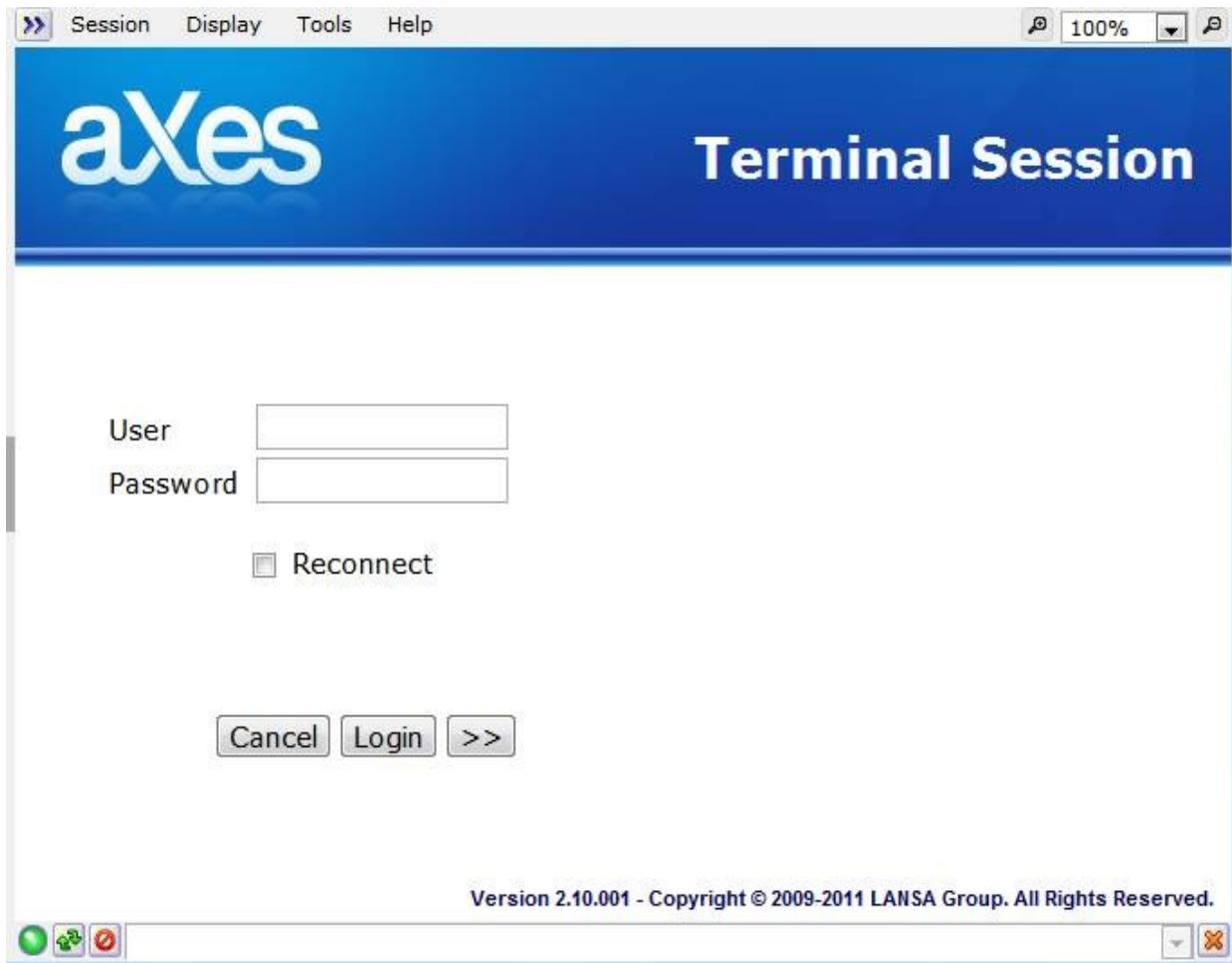
Remote server timeout: 30 Seconds ?

To set up your reverse proxy, replace MyHost:8080 with your host details. Once you've set up the Reverse Proxy you should:

- 1 Restart the web server
- 2 Clear the browser's cache

3. Start Fiddler!
4. Try first serving the equivalent of MyHost:8080/ts/ts2/index.html and then MyHost:81/ts/ts2/index.html.

MyHost:8080 is the RAMP-TS (aXes) domain therefore MyHost:8080/ts/ts2/index.html should work straight away and you should see a page like this:



If you cannot see this screen, you may not have aXes installed or there is a problem with your configuration.

MyHost:81 is the VLF domain. If you can serve the same page using the VLF domain it means the change to the web server configuration is working.

Configure RAMP

Perform these steps to configure RAMP in the Framework:

[If You Have Used RAMP-NL, Read This](#)

[Ensure Your Framework Has an Overall Visual Style Theme](#)

[Specify RAMP-TS Server Details](#)

[Set up Super-Server Session](#)

[Optionally Set up Framework Users and Security](#)

If You Have Used RAMP-NL, Read This

If you start up a Framework that is already using RAMP-NL, it will continue to use RAMP-NL. So to get started with RAMP-TS you need to start with a Framework that has no RAMP-NL details associated with it.

This means that the easiest way to get started with RAMP-TS is to:

Start VLF using the Latest Shipped Demonstration System (lastshipped.xml file) so you are using a default configuration.

You also need to ensure there is only one server defined and set it type “LANSA iSeries + RAMP TS”.

Remove any existing RAMP-NL server definitions.

Ensure the existing vf_sy001_nodes.xml file is renamed or removed.

Ensure you are using a unique set of XML file names to store your framework details in (on the Framework -> Properties -> Framework Details tab).

Use the “Save As” option to save this Framework with a different name.

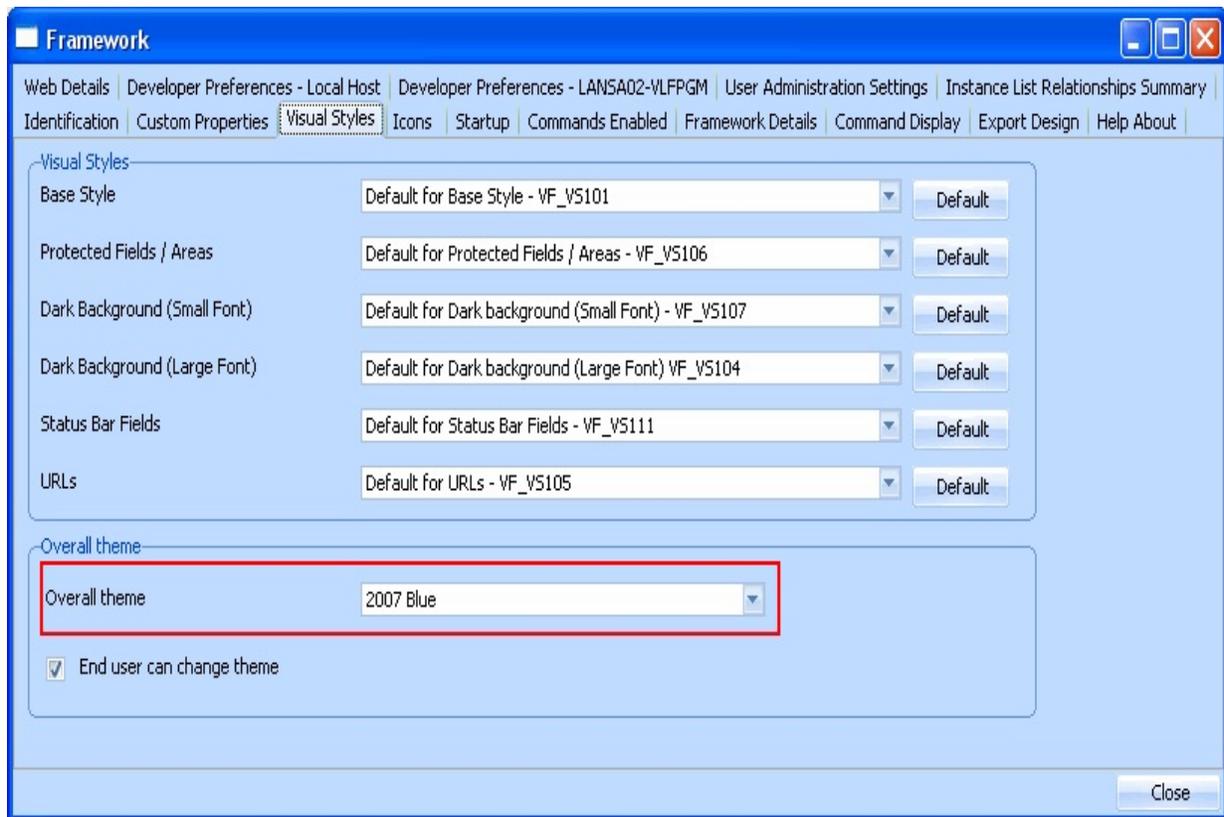
When you have successfully configured to use RAMP TS, when opening the RAMP Tools you should see this:



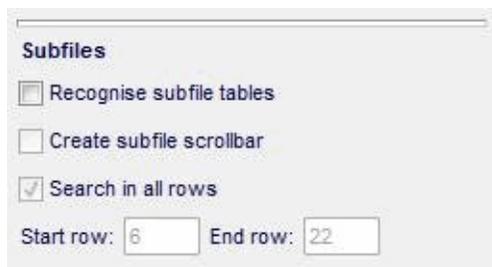
To understand more, see [How is a Framework associated with RAMP-TS or RAMP-NL?](#).

Ensure Your Framework Has an Overall Visual Style Theme

When using RAMP-TS you must use an overall theme for your Framework. The use of overall theme None is not valid in RAMP-TS applications.



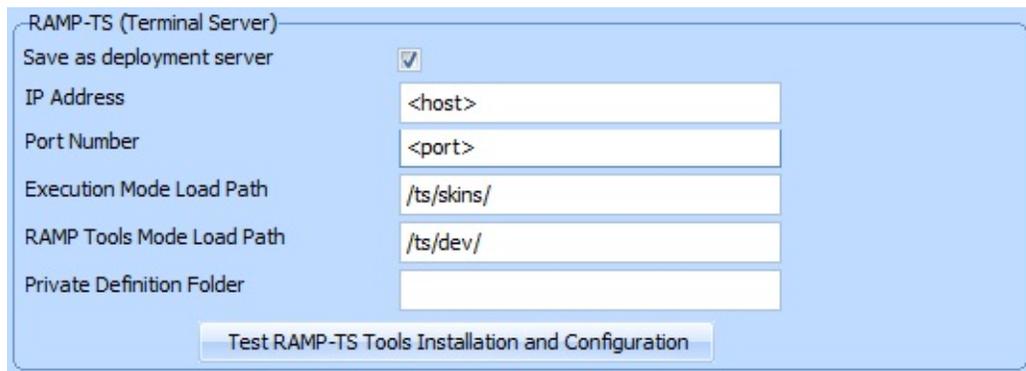
In aXes there are special autoGUI subfile handling options which must be turned off in RAMP:



They are not supported by RAMP-TS. In RAMP-TS many screens are hidden, in which case how subfiles are presented is irrelevant. Where a screen is made visible (ie: a destination) you can usually enhance the subfile in better way using eXtensions and other RAMP-TS techniques.

Specify RAMP-TS Server Details

1. Start the Framework as a Designer.
2. In the (Administration) menu of the Framework select the (Servers...) option.
3. Select the server named MY/AS400, or create a new server by clicking the New button.
4. On the Identification tab enter a Caption that describes your server.
5. On the Server Details tab, enter a Name for the server and select LANSA for System i + RAMP-TS as the Server Type.
6. In the RAMP-TS (Terminal Server) section of the screen enter the host and port number of your RAMP-TS/aXes-TS server like this:

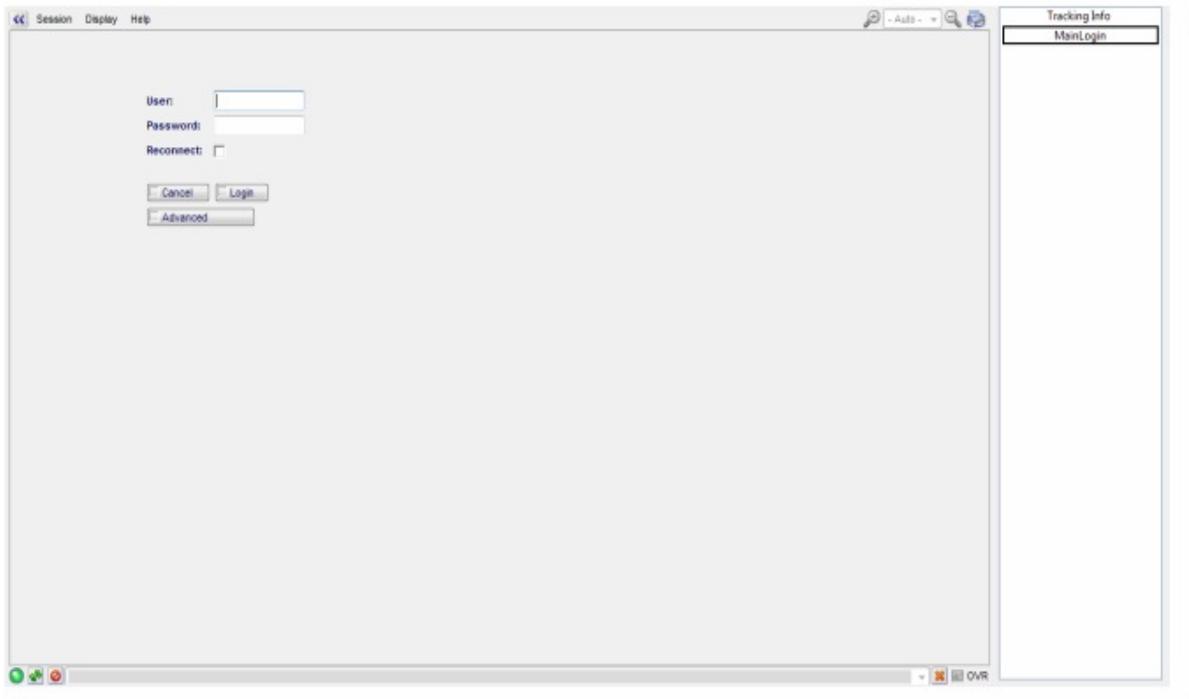


The screenshot shows a configuration window titled "RAMP-TS (Terminal Server)". It contains several fields and a checkbox:

- Save as deployment server:** A checkbox that is checked.
- IP Address:** A text input field containing the placeholder text "<host>".
- Port Number:** A text input field containing the placeholder text "<port>".
- Execution Mode Load Path:** A text input field containing the path "/ts/skins/".
- RAMP Tools Mode Load Path:** A text input field containing the path "/ts/dev/".
- Private Definition Folder:** An empty text input field.

At the bottom of the dialog is a button labeled "Test RAMP-TS Tools Installation and Configuration".

7. Save and restart the framework.
 8. Start the Framework as a Designer again. Use the (Administration) menu (Servers...) option again.
 9. Select the server you just defined and switch to the Server Details tab.
 10. Click the Test RAMP Tools Installation and Configuration button.
 11. On the resulting web sign on dialog sign on as user dev, password dev.
- The resulting screen should look like this:



Please do not proceed with using RAMP-TS until this verification test can be completed successfully.

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.

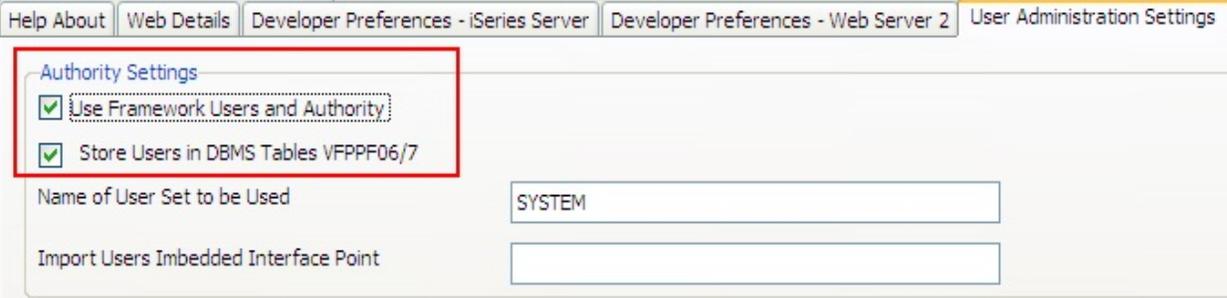
Close the dialog and save the Framework.

The screenshot shows the 'User Administration Settings' dialog box. The 'Sign on Settings' section is highlighted with a red box. The 'Users Sign on to a Remote Server to Use the Framework' option is selected, and the 'Users May Work Offline if the Remote Server Is Not Available' checkbox is checked. The 'Authority Settings' section includes options for 'Use Framework Users and Authority', 'Store Users in DBMS Tables VFPPF06/7', 'Store Users in XML File Named' (set to 'vf_Sy001_Users.xml'), and 'Import Users Imbedded Interface Point'. The 'Sign on Settings' section includes 'End Users must Signon to this Framework' (set to 'in both MS-Windows and Web Browser Applications'), 'Users Sign on Locally to Use the Framework' (unselected), 'User Can Change Own Password' (checked), 'Maximum Signon Attempts Allowed' (set to 'Advise User with a Message'), and 'Framework Fatal Error' (unselected).

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

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.

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](#)

[Script Naming Convention](#)

Handle Multiple Framework Versions

See [Framework Versions](#).

Script Naming Convention

RAMP scripts are assigned names like FORMSCRIPT_137. 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 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 [xxxxxxx is an orphan script and should be deleted](#).

Starting RAMP

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

[Start LANSAs](#)

[Start the Framework](#)

[Start RAMP](#)

[Start the Instant Prototyping Assistant](#)

[Start the Program Coding Assistant](#)

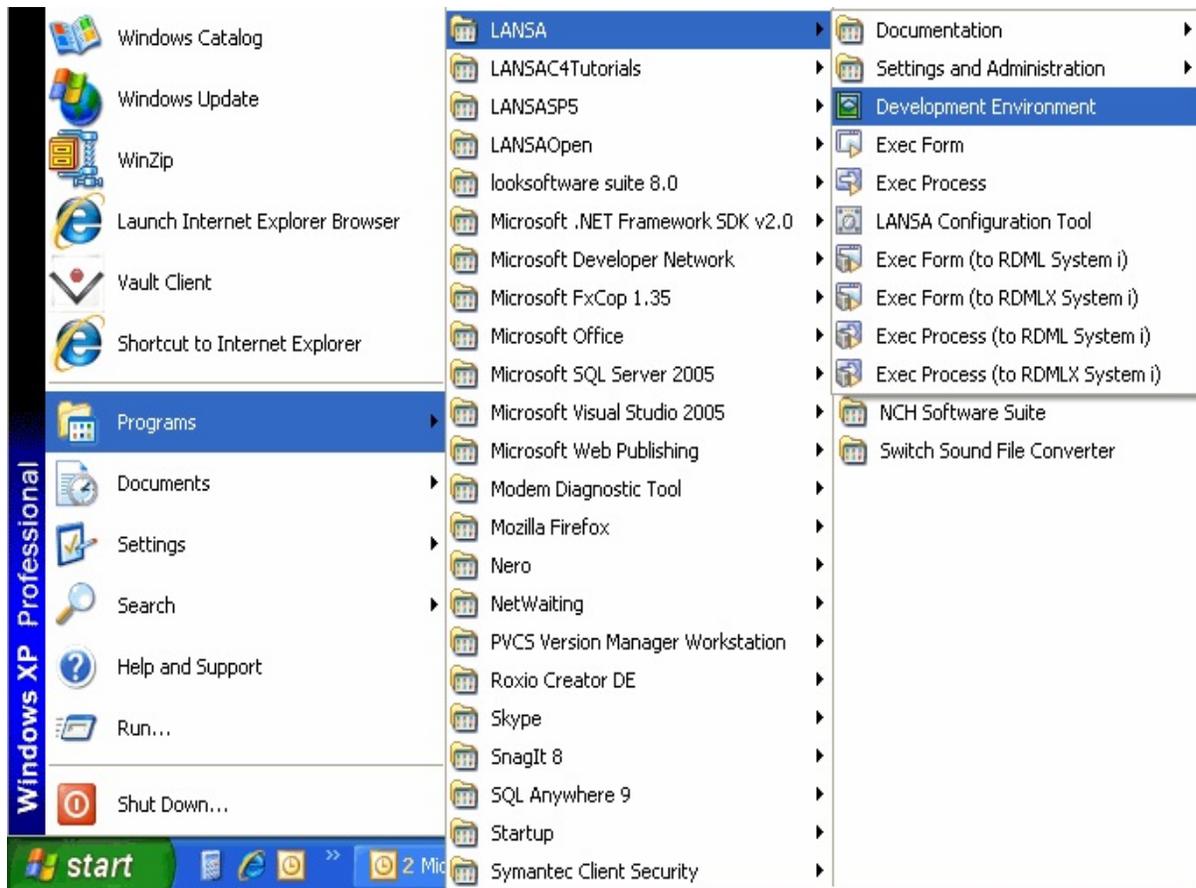
Start LANSAC

To start LANSAC:

Use the Start menu and display the Programs folder.

Select LANSAC.

Select the Development Environment option



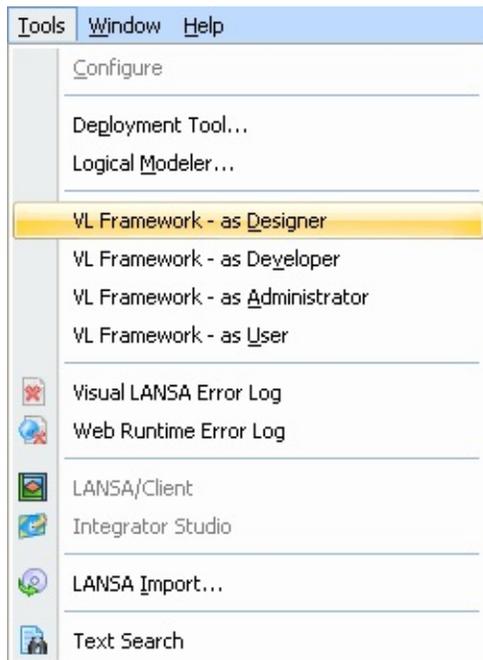
The LANSAC development environment is displayed

Start the Framework

You start the Framework from the LANSAs development environment:

Display the Tools menu.

Select the VL Framework - as Designer option.

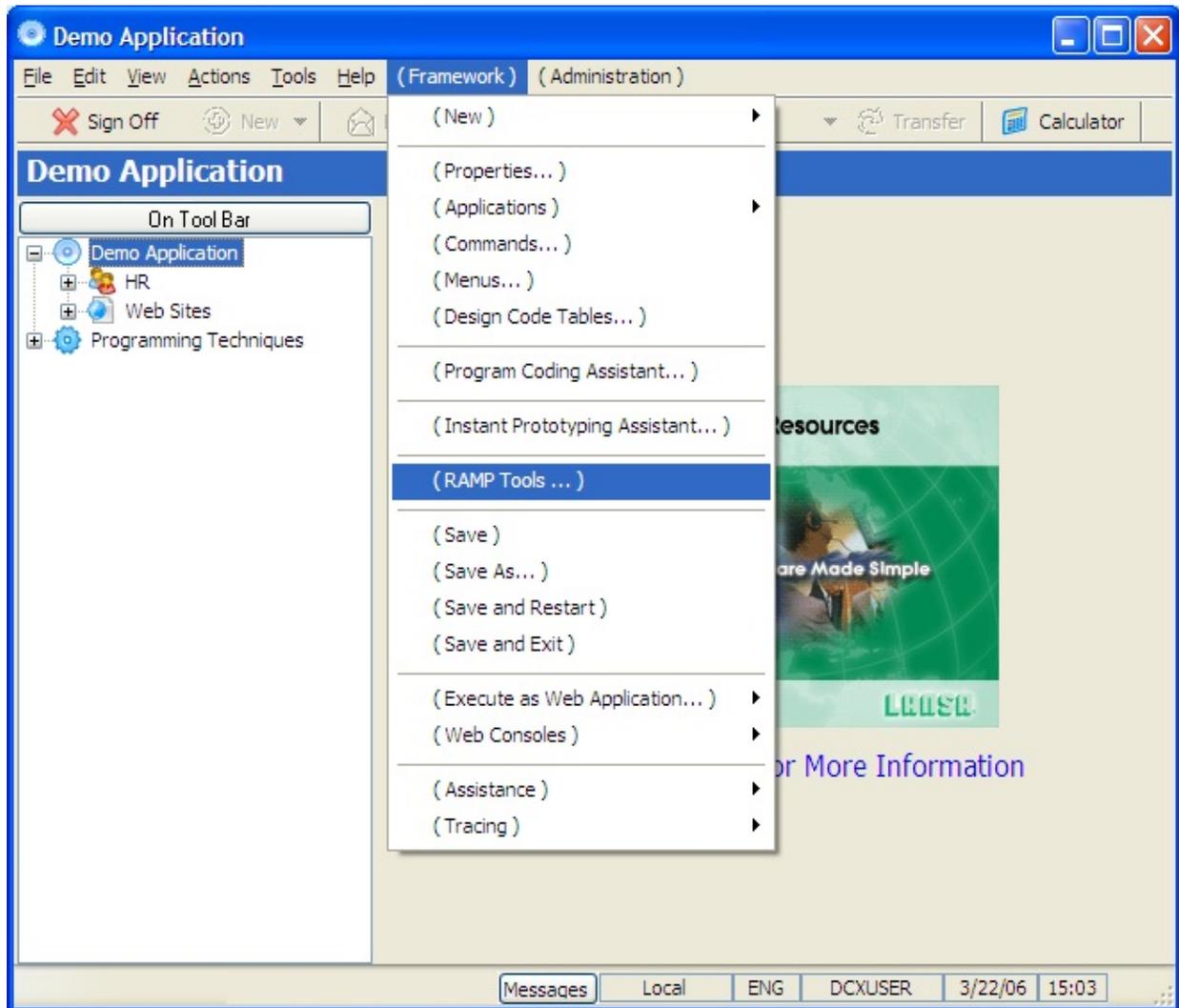


Start RAMP

You start RAMP from the Framework window:

Display the Framework menu.

Select the RAMP Tools... option.



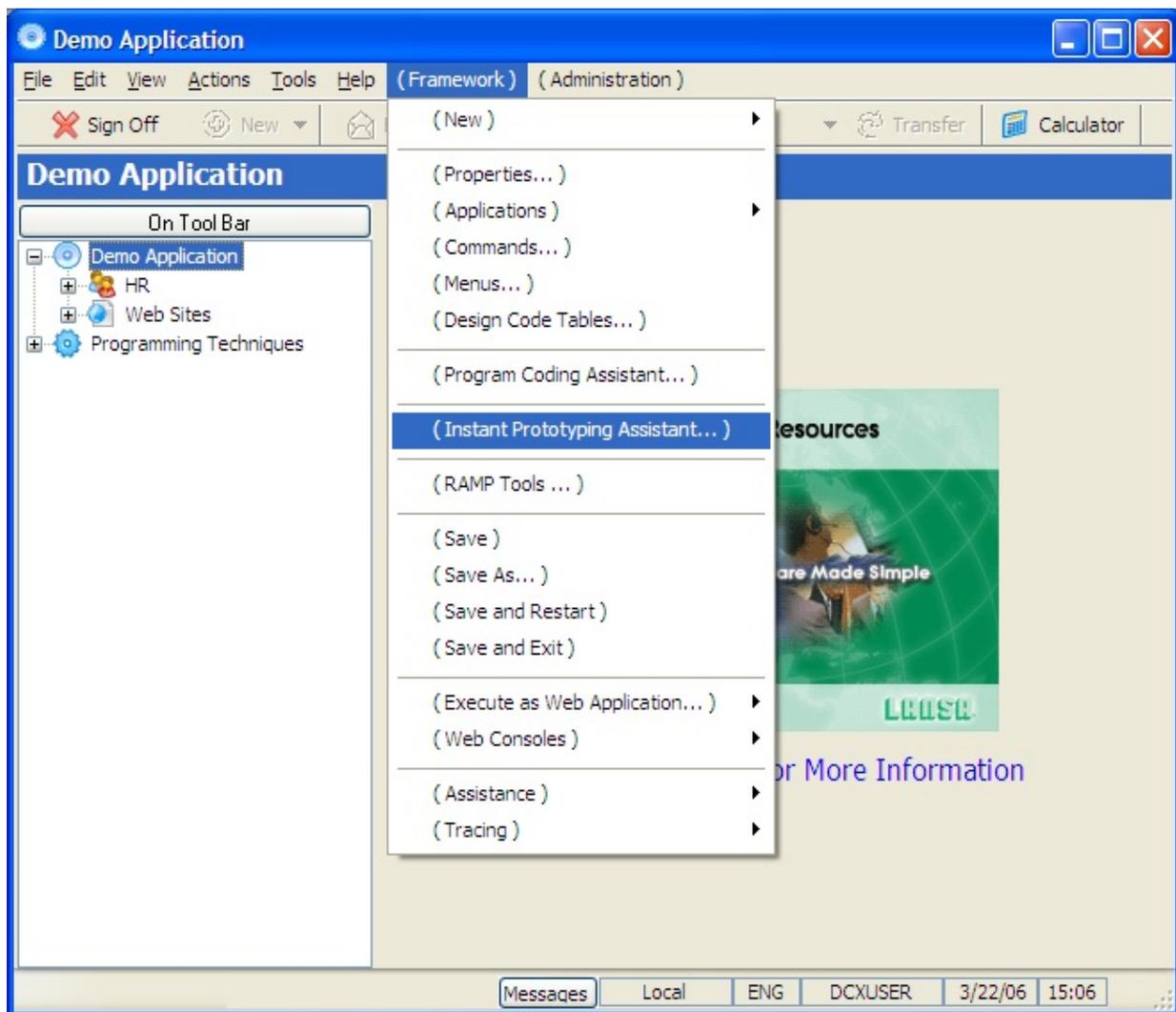
The RAMP Tools window is displayed.

Start the Instant Prototyping Assistant

The tutorial [RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#) 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:



And then respond Yes to the message that appears:



The Instant Prototyping Assistant is displayed:

Instant Prototype Assistant

define business objects ↔ attach actions ↔ put business objects in groups

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 known.
 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:
 Edit, Print, Delete, Accounts, Recent Transactions, Correspondence, Verify

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.

<ul style="list-style-type: none"> Details New Notes 		<ul style="list-style-type: none"> employee (exists already) <ul style="list-style-type: none"> Video Action: Video emp... Email Action: Email empl... Edit Action: Edit emplo...
---	---	--

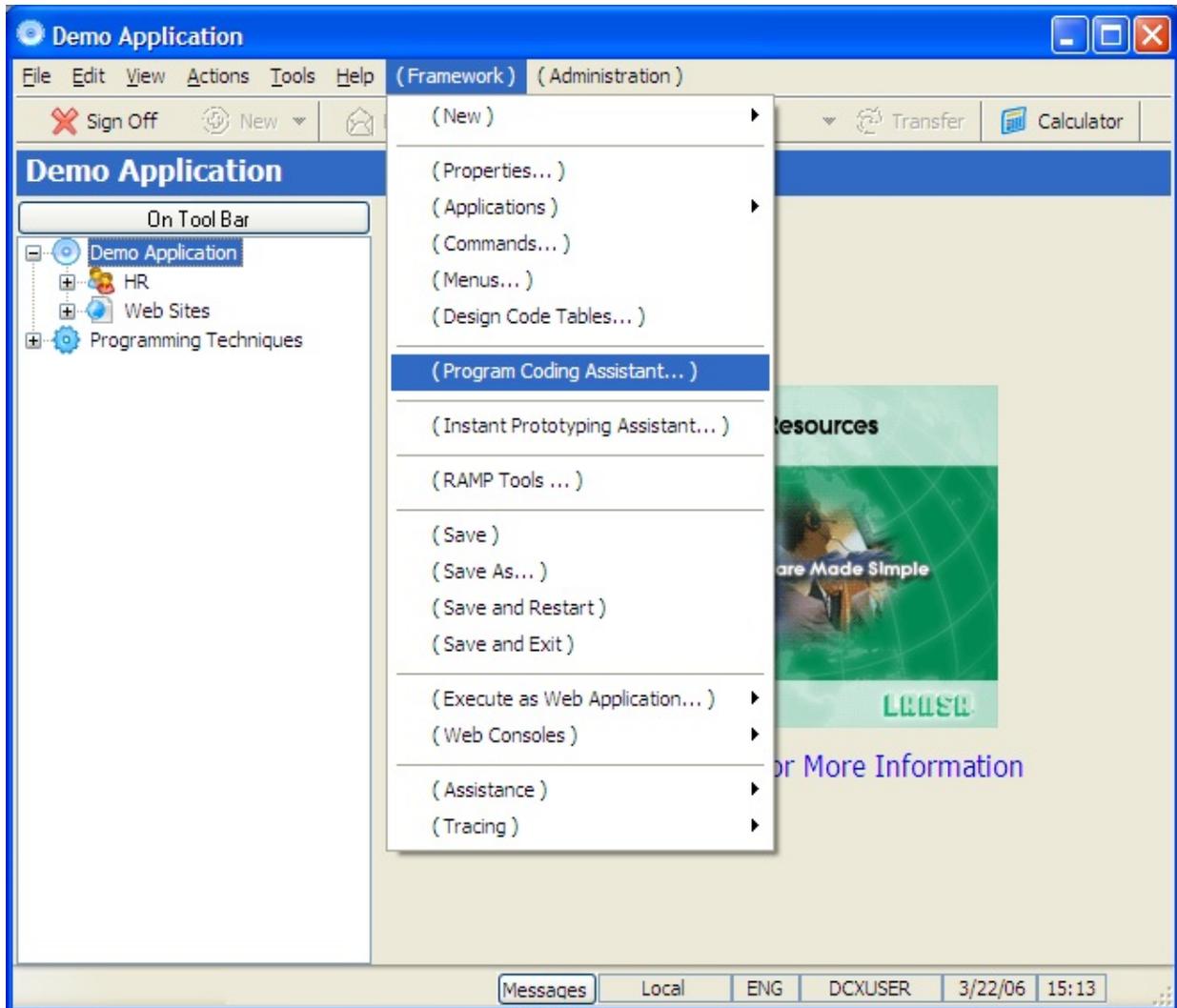
<< Back Next >> Cancel

Start the Program Coding Assistant

The tutorial [RAMP-TS003: Creating a Data Filter for Employees](#) 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

Naming Conventions

OBJECT-ACTION User Interfaces

Steps Involved in Using RAMP

You need to complete these steps:

Step	Comments	Modernization of Infrequently Used Application 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.		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 required Framework filters	Using powerful filters is the basis of reusing the 5250 screens in new modernized ways.	Not necessary.	
5. Optionally add new features making use of Windows	For example you may want to add advanced screens for for instance email, video,		

functionality

graphing.

6. Deploy

Deploy your
application

Framework Window

Navigation Pane
Use this area to move between applications and business objects

Filters
Use filters to specify selection criteria for the objects to work with

Instance List
This list shows the objects that meet the filtering criteria. Select from it the object you want to work with.

Number	Name	Phone
A3564	FREDDY BROWN	(02) 567-6758
A0090	FRED JOHN ALAN...	344-2234454545
A0070	VERONICA BROWN	(02) 9609 4627
A1031	JOHN BLAKE	(02) 9668 9235

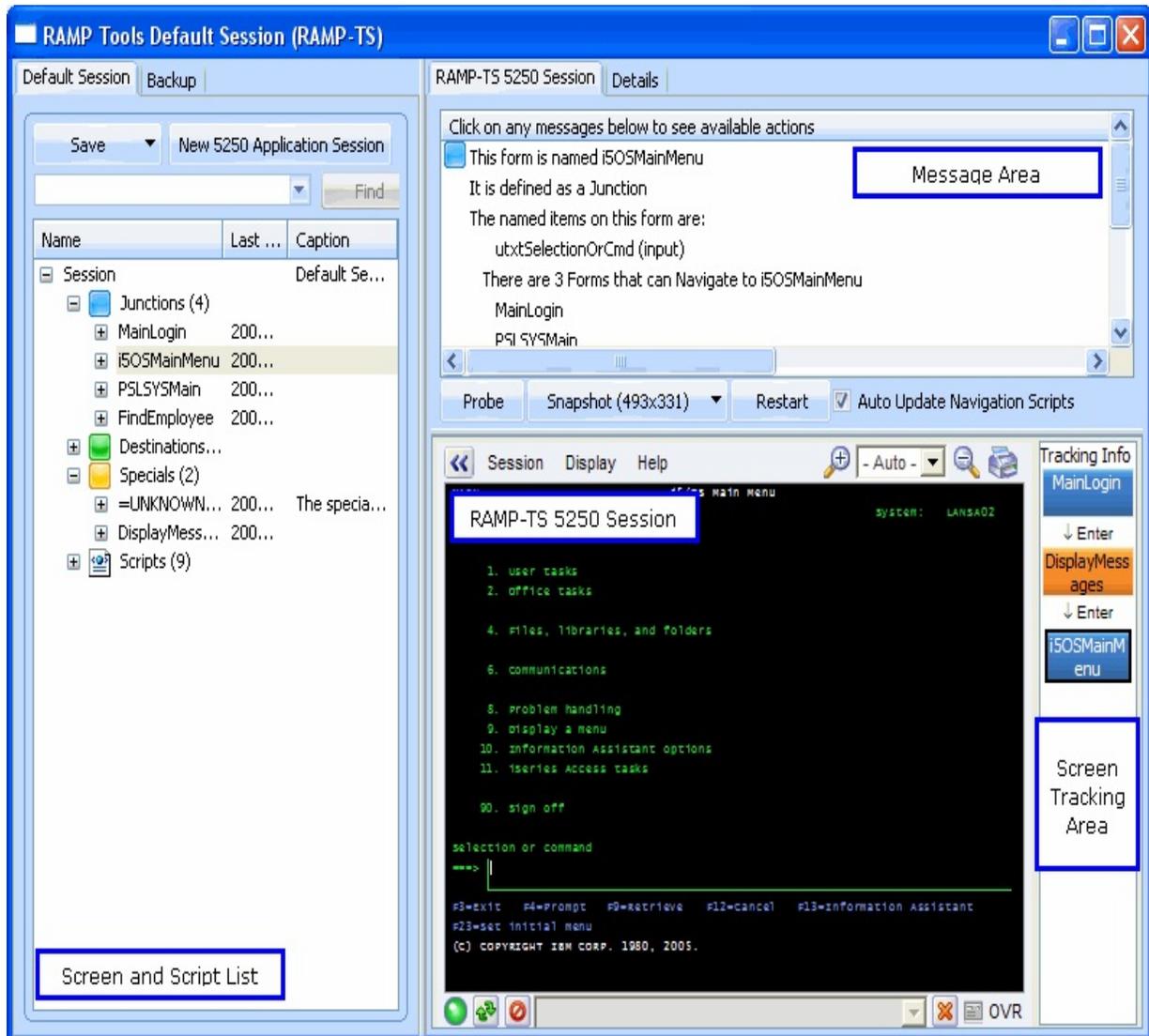
Command Handlers
The programs you use to work with objects. They can be RAMP iSeries screens or LANSA components.

Employee : Basic details (A0070-VERONICA BROWN)

Employee Number	A0070	Save
Employee Surname	BROWN	
Employee Given Name(s)	VERONICA	
Street No and Name	12 Railway Street	
Suburb or Town	Baulkham Hills	
State and Country	NSW Australia	
Post / Zip Code	2153	
Home Phone Number	(02) 9609 4627	

Messages | Ready | Local | ENG | DCXUSER | 10/10/06 | 15:42

RAMP Window



Message Area

Screen Tracking Area

RAMP-TS 5250 Emulator Session

Screen and Script List

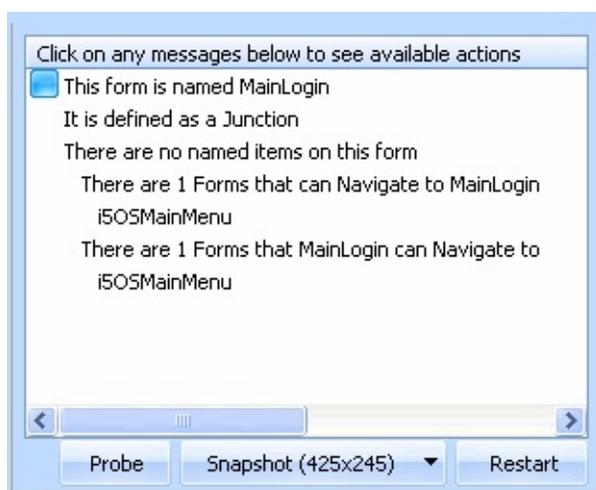
Details Area

Message Area

RAMP Window	Screen Tracking Area	RAMP-TS 5250 Emulator Session	Screen and Script List	Details Area
-----------------------------	--------------------------------------	---	--	------------------------------

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

Messages are shown for the screen selected in the [Screen Tracking Area](#). You use this area also to classify screens:



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: <ul style="list-style-type: none"> • 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

	<ul style="list-style-type: none"> • 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.
Auto update Navigation Scripts	Use this option to turn the automatic generation of navigation scripts on or off. When this option is selected, a red indicator light is shown next to the check box.

Screen Tracking Area

RAMP Window	Message Area	RAMP-TS 5250 Emulator Session	Screen and Script List	Details Area
-------------	--------------	-------------------------------	------------------------	--------------

The Tracking area displays the screens you have displayed in the current RAMP-TS session. When you end the session, the tracking information is cleared.

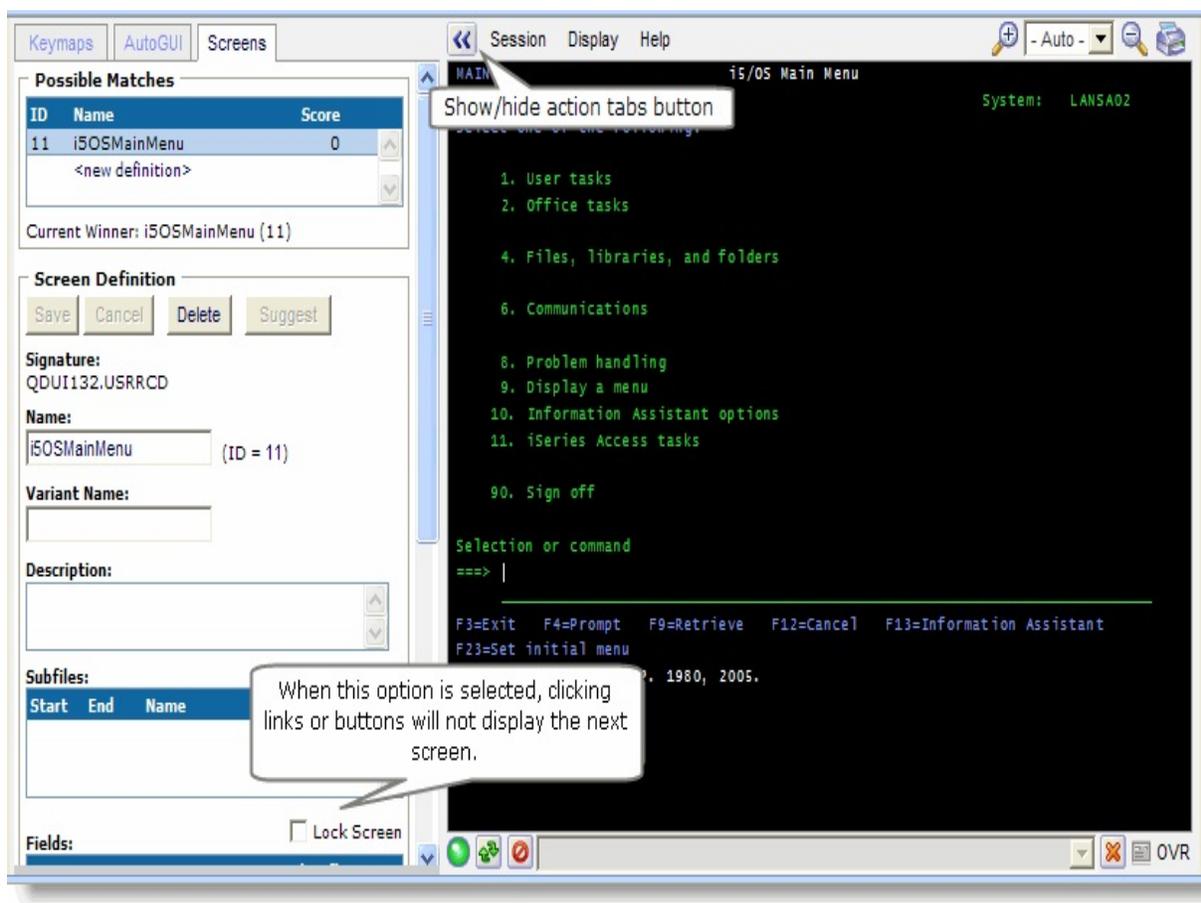
Colors used in the Tracking area for screens indicate their status and type you have assigned to the screen:

	The screen has not been identified.
	The screen has been named, but it has not been defined in RAMP. You need to define the screens in your application according to their purpose: <ul style="list-style-type: none"> • Destination Screens are screens where the end-user works • Junction Screens are used for navigation only • Special Screens are used for messages etc.
	The screen is a junction screen.
	The screen is a special screen.
	The screen is a destination screen.

RAMP-TS 5250 Emulator Session

[RAMP Window](#) [Message Area](#) [Screen Tracking Area](#) [Screen and Script List](#) [Details Area](#)

The RAMP-TS emulator session shows the 5250 session screens with action tabs shown on the left. The Screens tab is where you specify the name of the screen and any fields you need to name:



When the Lock Screen option is selected, the current screen will remain displayed when you click on links or buttons that would bring up other screens. You can hide the action tabs by clicking on the Show/Hide button:

```
Session Display Help
MAIN                               i5/OS Main Menu                               System:  LANS02
Select one of the following:

1. User tasks
2. Office tasks

4. Files, libraries, and folders

6. Communications

8. Problem handling
9. Display a menu
10. Information Assistant options
11. iSeries Access tasks

90. Sign off

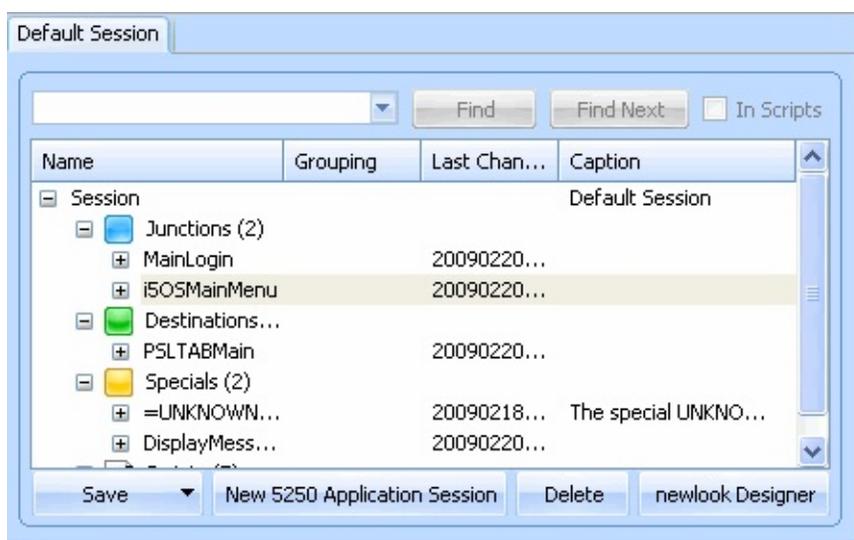
Selection or command
==> |
-----|
F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=Information Assistant
F23=Set initial menu
(C) COPYRIGHT IBM CORP. 1980, 2005.
```

You should use the menus(Session, Display and Help) in the RAMP-TS session window only when asked by your product vendor.

Screen and Script List

RAMP Window	Message Area	Screen Tracking Area	RAMP-TS 5250 Emulator Session	Details Area
-------------	--------------	----------------------	-------------------------------	--------------

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



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	<p>Performs a full Framework save including the generation of all scripts for execution in end-user mode and the uploading of web server details.</p> <p>You would normally do a full save before you want to test your application in end-user mode.</p>
Partial Save	<p>Performs a partial Framework save so that your work is fully recoverable, but does not generate run-time scripts or upload server</p>

details.

You will need to do a full Framework save to deploy your application or execute it in end-user mode.

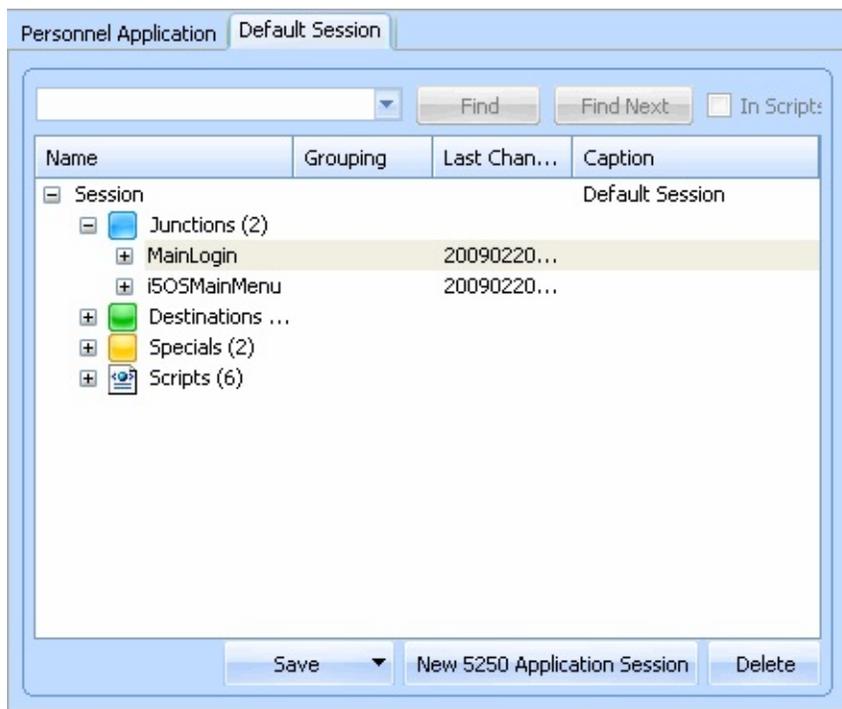
You can use the New 5250 Application Session button to organize screens and scripts into distinct 5250 Application Sessions (see [Organizing Screens and Scripts](#)).

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 a Personnel Application session has been created in addition to the Default Session:



To create a new application session, click the New 5250 Application Session button. To delete an application session, select Session in the screen and script list and click Delete.

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 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 Area](#)

Session Details

RAMP Window	Message Area	Screen Tracking Area	RAMP-TS 5250 Emulator Session	Screen and Script List
-------------	--------------	----------------------	-------------------------------	------------------------

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

RAMP-TS 5250 Session Details

Session - Default Session

Caption

User Object Name / Type

Default RAMP Layout Dimensions

Height Width Top Left

Top Mask Height Bottom Mask Height

RAMP Screen Layout Style

Fixed Layout Flow Layout

Scroll Bars

Display Horizontal Scroll Bars Display Vertical Scroll Bars

Lock framework when unknown 5250 form is displayed

Reuse existing connection's user profile and/or password

Always link this session to a server with User Object Name / Type

RAMP-TS Maximum Logoff Wait Time (seconds)

Special Field Handling

	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)
1	DEPTMENT	F4	DF_PRM04
2	SECTION	F4	DF_PRM04
3	DATE_*	F4	DF_PRM07
4		F4	
5		F4	

Caption	The caption of the RAMP 5250 Application Session.
---------	---

User Object Name / Type	See Object Type in the Framework Guide.
Height	The default height of 5250 screens when displayed in the Framework.
Width	The default width of 5250 screens when displayed in the Framework.
Top	The default distance between the top of the RAMP screen tab and the 5250 screen. You can use this option to Hide screen titles in individual 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 individual RAMP Screens 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 Style	If RAMP Screen Layout Style is set to Flow, RAMP screens will be automatically resized to fit into the space available to display them. If Flow is used: <ul style="list-style-type: none"> • 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. Fixed means the RAMP screens are not resized to fit into the

	space available to display them.
Scroll Bars	<p>If the Display Horizontal Scroll Bars 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 Display Horizontal Scroll Bars option is checked.</p> <p>If the Display Vertical Scroll Bars 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 Display Vertical Scroll Bars option is checked.</p>
Lock Framework when unknown 5250 form is displayed	<p>This option applies a lock to the Framework when an unknown 5250 screen is encountered.</p> <p>When a lock is applied, the user cannot move around within the Framework until they navigate to a defined 5250 screen.</p> <p>They can exit from (for example, shut down) the Framework when such a lock has been applied.</p> <p>Typically this option is used to trap unknown and/or unexpected 5250 screens.</p> <p>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.</p>
Reuse existing connections user profile and/or password	<p>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.</p> <p>This option can 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.</p> <p>This option can be automatically overridden by individual user profile options or by super-server connection values.</p>

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.

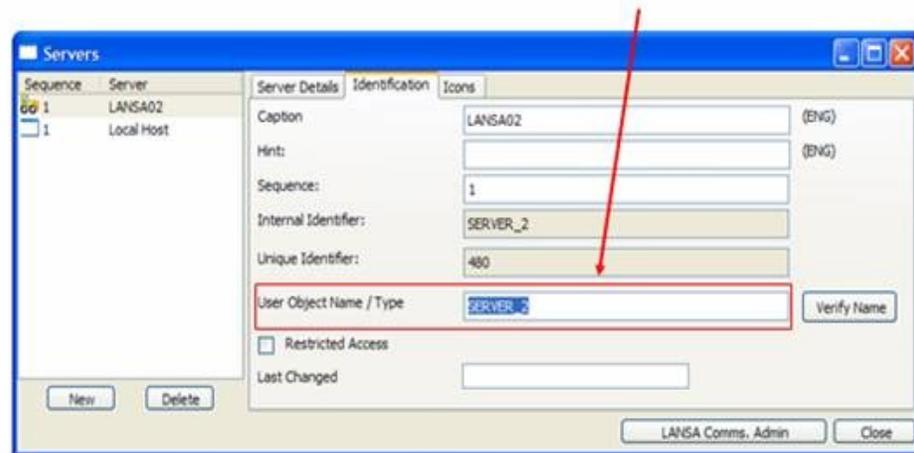
Use this option to prevent the user from having to, or being allowed to, making this server connection choice.

Using it unconditionally links a 5250 application session with a server.

To use this option first assign a 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.



Next, set the 5150 application session to use the same name (eg: SERVER_2).

Now the 5250 application session and the server with user object name/type SERVER_2 are unconditionally linked.

The user 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.

RAMP-TS Maximum Logoff Wait Time (seconds)	When a RAMP-TS session needs to be logged off (signed off) this values specifies in seconds the maximum time that the framework should wait for asynchronous time log off operation to complete. The default value is 10 seconds. Specify any integral value in the range 0 to 120. The value is in seconds.
Special Field Handling	Advanced prompting facility for fields. You specify the name of the field to be prompted, the function key to be used and the Visual LANSA form that is used as the prompter. For more information refer to Advanced Prompting

Destination Screen Details

RAMP Window	Message Area	Screen Tracking Area	RAMP-TS 5250 Emulator Session	Screen and Script List
-------------	--------------	----------------------	-------------------------------	------------------------

When a [Destination Screen](#) is selected in the [Screen and Script List](#), the details of the destination screen are shown:

The screenshot shows the PSLTABMain configuration window with the following sections:

- Caption:** A text input field.
- Grouping:** A text input field.
- Default newlook Layout Dimensions:**
 - Height: 412
 - Width: 710
 - Top: [input]
 - Left: [input]
 - Top Mask Height: [input]
 - Bottom Mask Height: [input]
- RAMP Screen Layout Style:**
 - Session
 - Fixed Layout
 - Flow Layout
- Targets:** A list box containing "ISOSMainMenu" with a "Save" button below it.
- Targeted By:** A list box containing "ISOSMainMenu" with a "Save" button below it.
- Function Key Enablement:** A table with columns: Key, Caption, Enable 5, Enable, Seq.

Key	Caption	Enable 5	Enable	Seq
Enter	Enter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2
F2	F2	<input type="checkbox"/>	<input type="checkbox"/>	3
F3	Exit	<input type="checkbox"/>	<input type="checkbox"/>	4
F4	F4	<input type="checkbox"/>	<input type="checkbox"/>	5
F5	F5	<input type="checkbox"/>	<input type="checkbox"/>	6
F6	F6	<input type="checkbox"/>	<input type="checkbox"/>	7
F7	F7	<input type="checkbox"/>	<input type="checkbox"/>	8
F8	F8	<input type="checkbox"/>	<input type="checkbox"/>	9
- Associated Command Handler(s):** A table with columns: Link to Command (Tab), in Object, User Object T...

Link to Command (Tab)	in Object	User Object T...
<input type="checkbox"/> Details	TS_Test	TS_TEST
<input type="checkbox"/> New	TS_Test	TS_TEST
<input type="checkbox"/> Notes	TS_Test	TS_TEST
<input checked="" type="checkbox"/> Details	Personnel Tables	PERSONNEL_...

 Below the table is a "Refresh" button.
- Session Id:** A dropdown menu with the value "*AUTO".

You can specify these details for the destination screen:

Caption	

Grouping	<p>Optionally type a grouping name for this screen.</p> <p>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.</p> <p>For more fundamental organization of screens and scripts, see Organizing Screens and Scripts.</p>
Default RAMP Layout Dimensions	<p>Use these properties if you want to permanently override the default layout dimensions set in Session Details for this screen.</p>
RAMP Screen Layout Style	<p>If RAMP Screen Layout Style is set to Flow, RAMP screens will be automatically resized to fit into the space available to display them.</p> <p>If Flow is used:</p> <ul style="list-style-type: none"> • 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. <p>Fixed means the RAMP screens are not resized to fit into the space available to display them.</p> <p>Session means the value is inherited from the Session's properties.</p>
Targets	<p>This list shows the screens this screen can navigate to.</p> <p>The first screen in the list is the exit junction, that is, the screen to which this screen navigates to by default. You can override the exit junction in your script using the <code>vOverrideExitJunction</code> property.</p>
Targeted By	<p>This list shows the screens that can navigate to this screen.</p>
Function	<p>This is a list of all the available function keys in 5250 screens.</p>

Key Enablement	<p>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.</p> <p>Note that function key enabling is only valid for those function keys already present in the 5250 screen.</p> <p>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.</p> <ul style="list-style-type: none"> • To enable a function key, tick the check box in the Enable Key column. • To display the function key as a button, tick the check box in the Enable Button column. • The captions of the buttons can be changed in the Caption column. <p>The function keys and buttons can be overridden at execution time using the SETKEYENABLED Function.</p>
Associated Command Handlers	<p>The command handler tab where the RAMP screen will be attached.</p> <p>The command handler tabs are created when you prototype your application.</p>
Session ID	<p>Specifies what System i 5250 session (ie: job) should be started for the screen.</p> <p>*AUTO : is the default value and indicates that the Framework should manage the required 5250 session(s) automatically. This type of session is a managed session. It is fully integrated with the Framework, applications, business objects and instance lists and all scripting facilities are available.</p> <p>SESSION_A -> SESSION_Z: 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</p>

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.

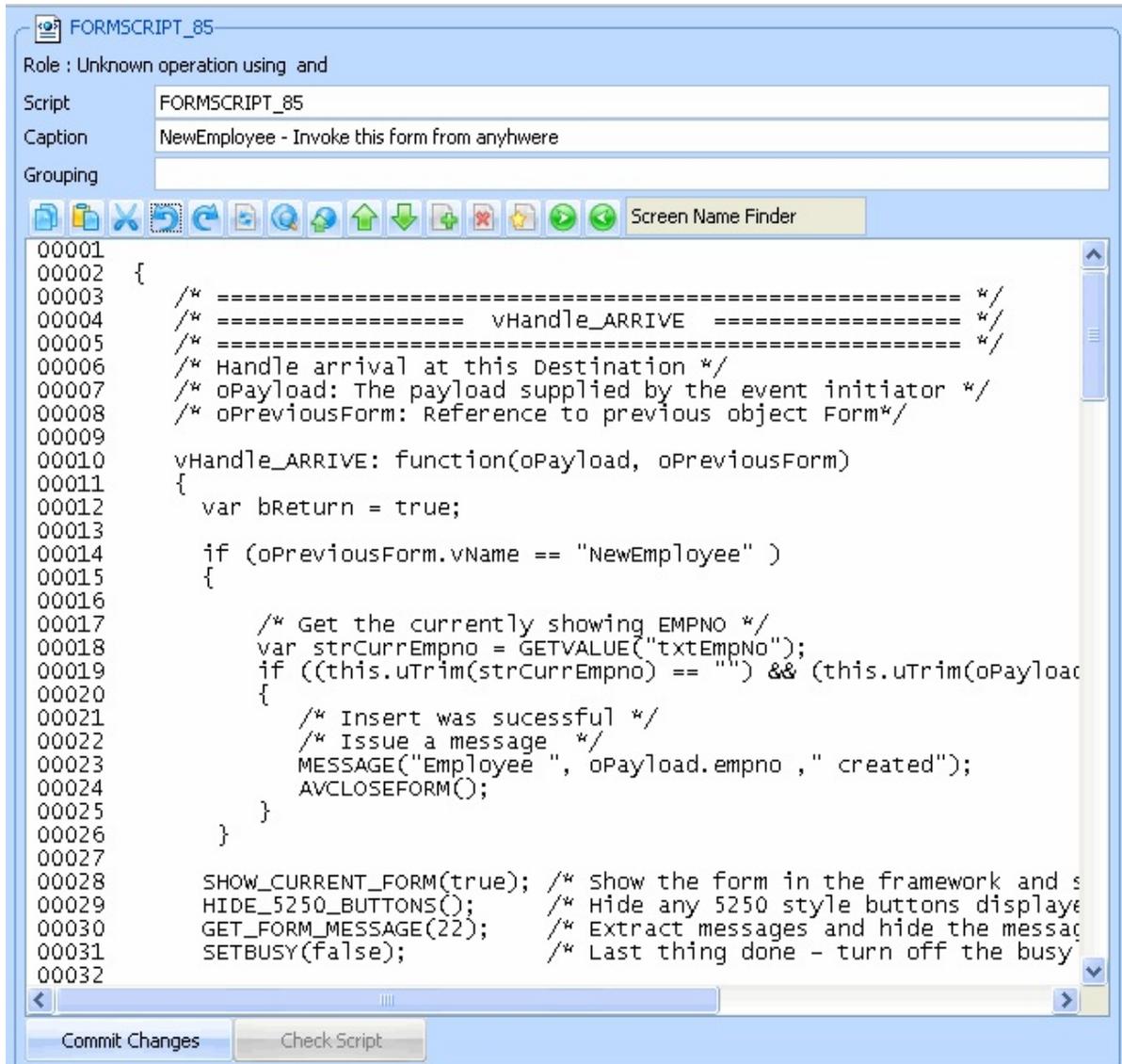
In short, you should always use *AUTO unless you have a specific need to log a user on, drive them a defined starting point in the application, and then allow them to move around wherever they like within the 5250 application area.

Script Area

RAMP Window	Message Area	Screen Tracking Area	RAMP-TS 5250 Emulator Session	Screen and Script List
-------------	--------------	----------------------	-------------------------------	------------------------

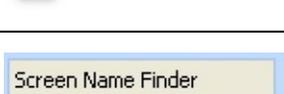
The Details area shows the details of the script selected in the [Screen and Script List](#).

The scripts are generated automatically as you track your application, but sometimes it is necessary to edit the scripts.



The scripting toolbar has buttons to assist you when working with scripts. From left the buttons are:

	Copy
	Paste
	Cut

	Undo
	Redo
	Select All
	Find
	Replace
	Increase Font Size
	Reduce font size
	Show/Hide line numbers
	Comment out lines
	Uncomment out lines
	Indent lines
	Unindent lines
	Print Code
	Commit Changes
	Check Script
	

which you can use to rapidly locate a screen name, click on name to get the name of the screen pasted into your code at current insertion point.

You can use the [Scripting Pop-up Menu](#) to help you to format and edit your scripts.

Screen Name Finder

When writing RAMP-TS scripts you often need to type in long and case sensitive screen names. To make this process easier you can use the screen name finder which appears at the top of the script editor frame like this:



To use it, type in your script up to the point you need to input the screen name. For example:

```
if (oPreviousForm.vName == I  
{
```

With the editor insert point caret where you want the screen name to be inserted. Move up and click in the screen name finder. Use it to rapidly locate the name of the screen you want to use by typing in a string that would be found in its name or caption (for example emp) and then select the screen from the list that is displayed.

The screen name, surrounded by double quotes will be pasted into your script, like this:

```
if (oPreviousForm.vName == "DisplayEmployee"  
{
```

To cancel the screen name finder without selecting a screen name, click back into your script.

Types of Screens

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

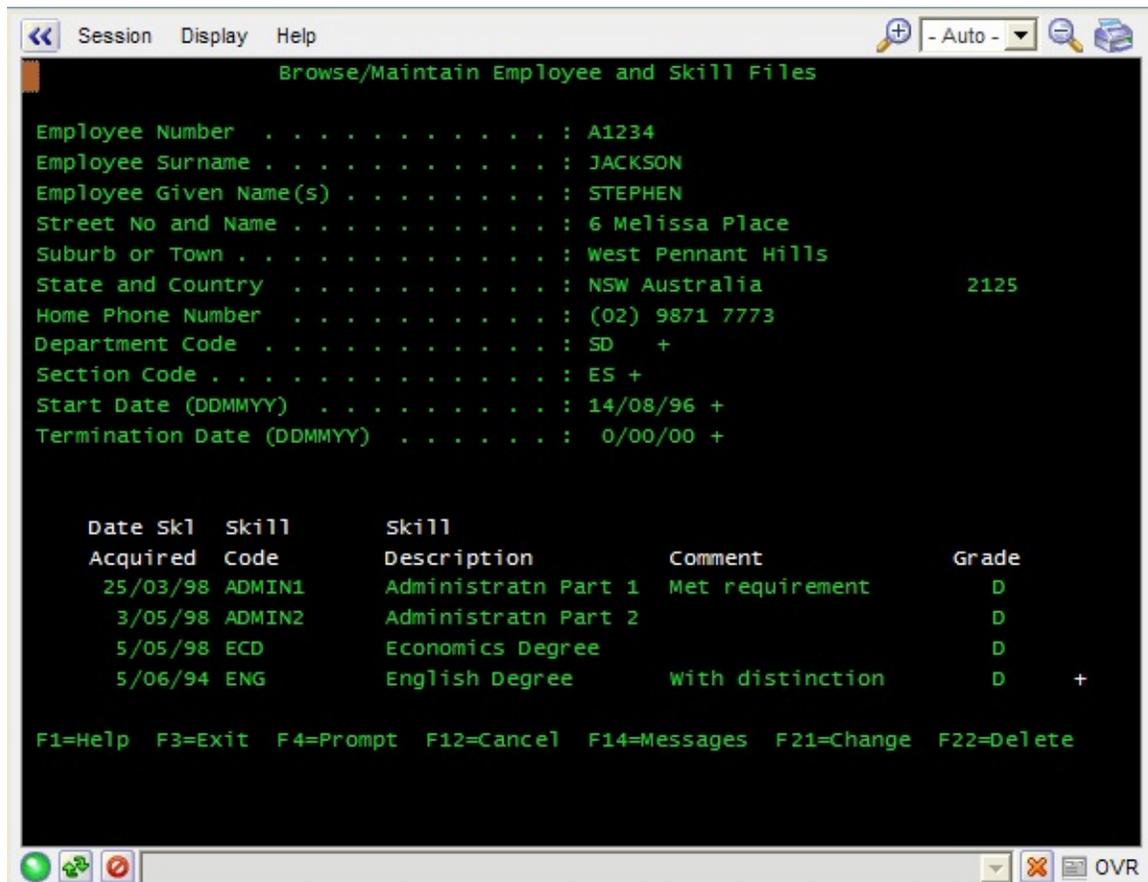
A **Destination Screen** is the 5250 screen where the end-user performs actual work. These screens are snapped into the Visual LANSA Framework without any modification.

A **Junction Screen** is used for navigation only. They are hidden in your modernized application.

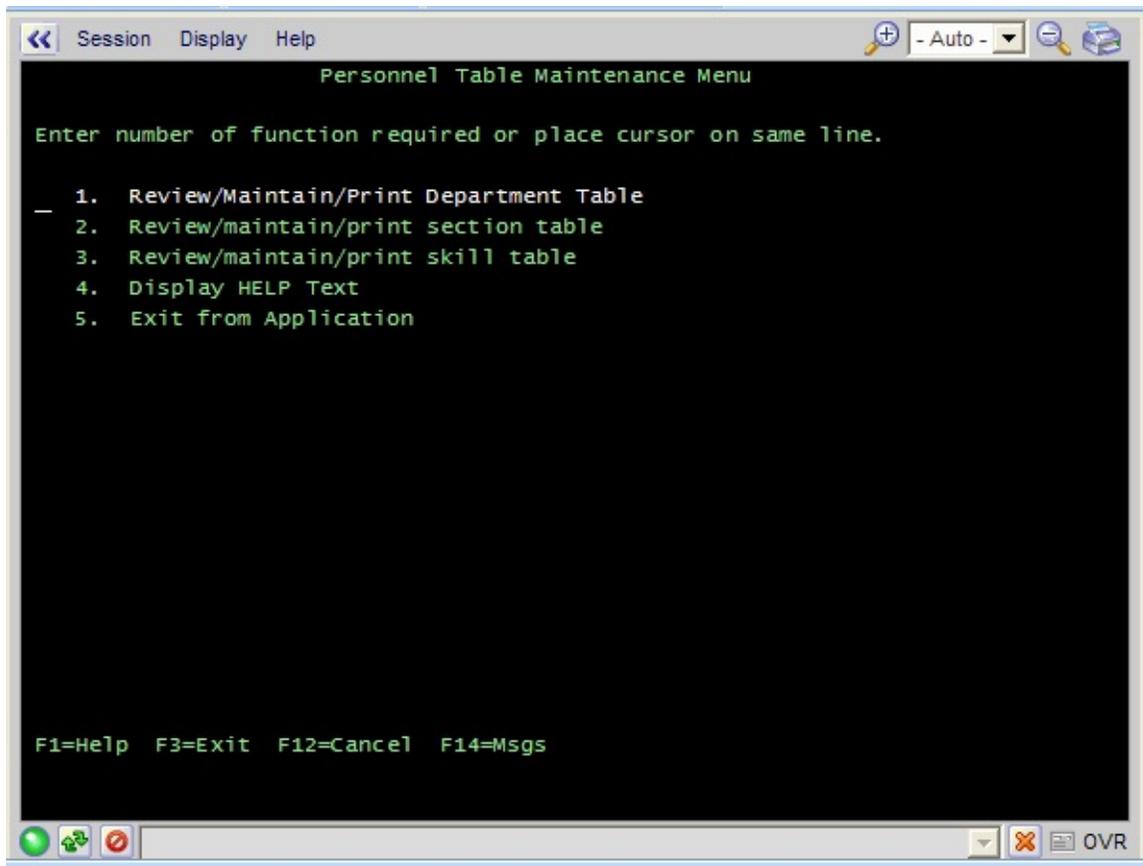
A **Special Screen** is a messages or other similar screen that does not fit the above two categories.

Destination Screen

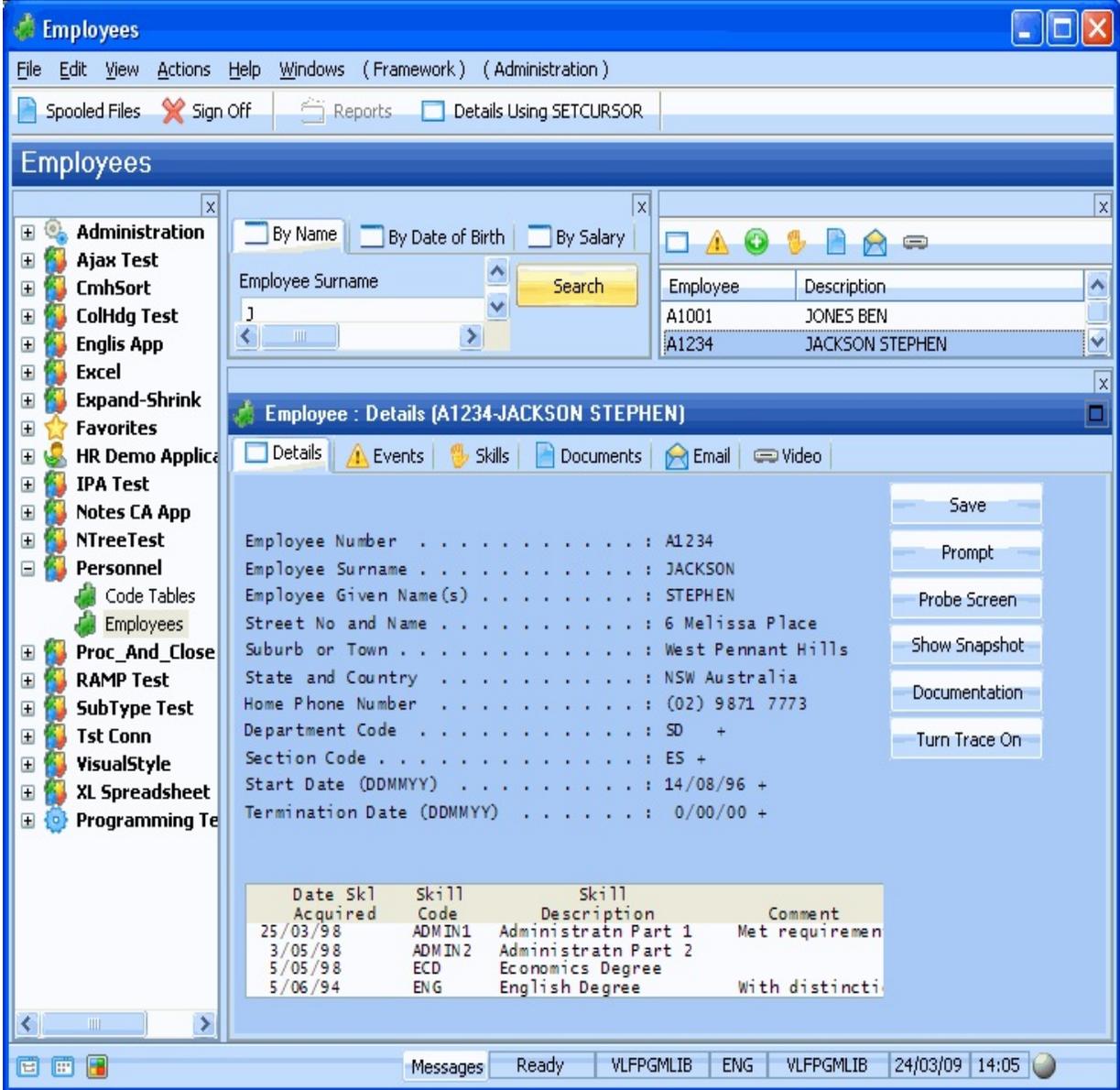
A destination screen is a screen in which the end-user works with an object. Typically it is a maintenance screen:



But it can also be a menu (see [RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)):

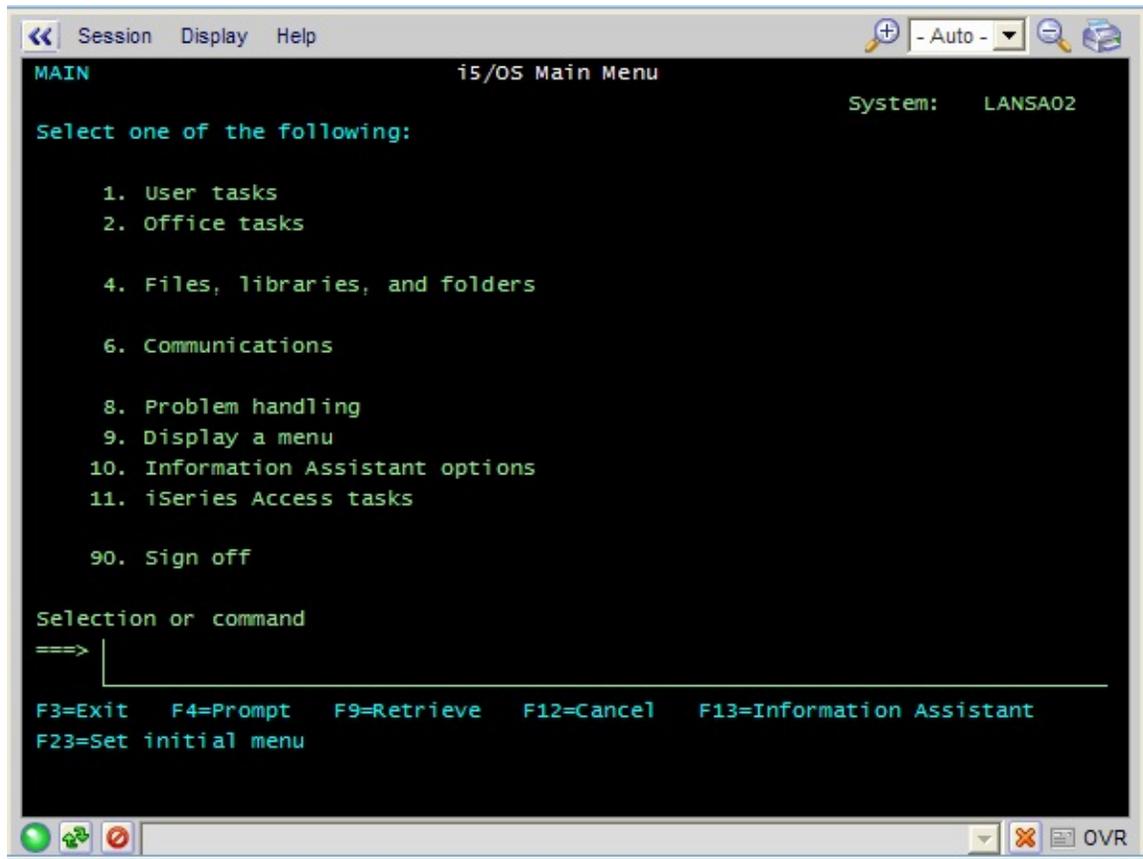


Destination screens can be reused without any modification in RAMP applications:



Junction Screen

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



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

Typical junctions are:

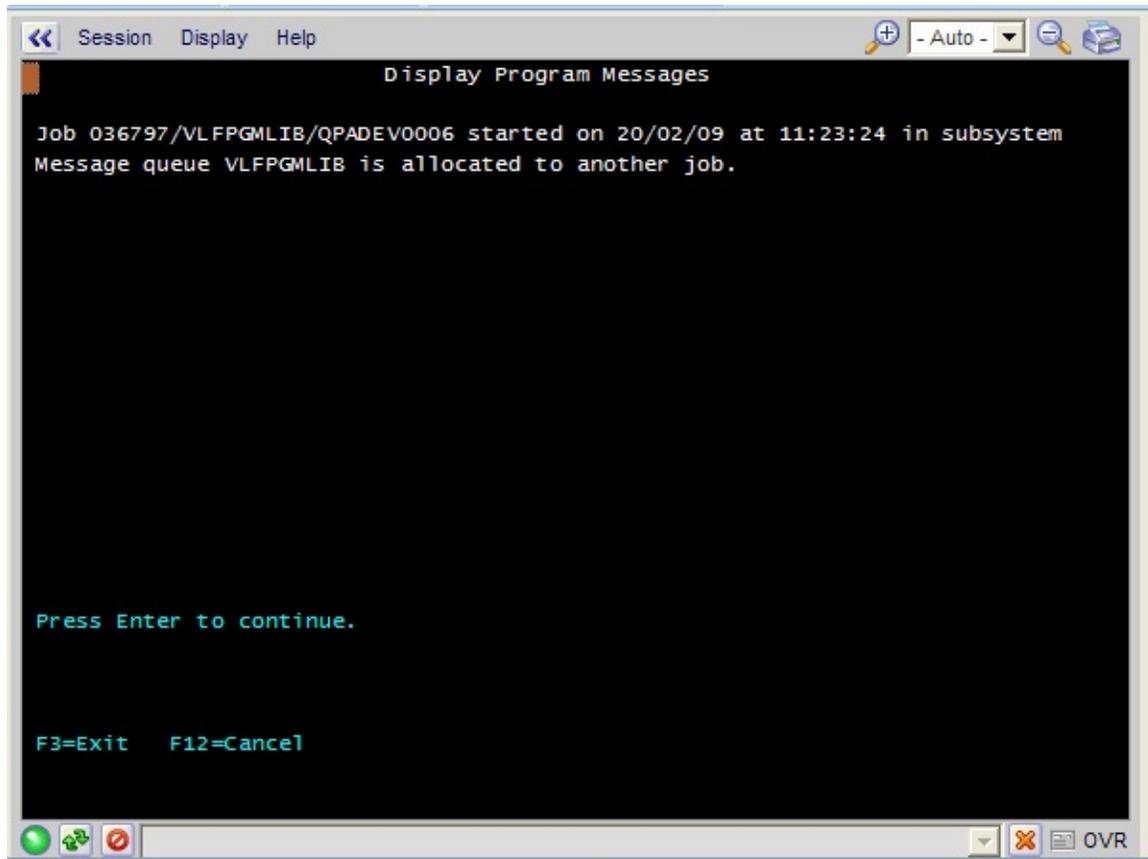
5250 menus

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.

Naming Conventions

The following standards are very strongly recommended for RAMP-TS projects:

Use characters from the English alphabet (A->Z, a-z) and numbers (0 – 9) only in names. The names are then code page invariant.

Never use imbedded blanks in names.

Limit name length to around 30 characters or less. Long names can be confusing and are error prone when scripting and debugging.

Generally names are case sensitive.

Never implement a case based naming standard that uses duplicate names. For example, a naming standard that used "EmployeeNumber" for a field on a screen and "EMPLOYEENUMBER" for the same field in a subfile on that screen will surely end up in a tangle at some stage. Names should be unique, regardless of their case.

Naming Prompters

RAMP-TS allows prompters to be automatically associated with named fields on 5250 screens.

When these prompters are defined, they may associate themselves with screen fields by using a specific name like StartDate or generic name like Date_* (meaning that any field whose name starts with Date_ should be associated with this prompter).

If you intend to use the automatic prompt capability with generic names it will probably influence your chosen naming standard.

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 an 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:

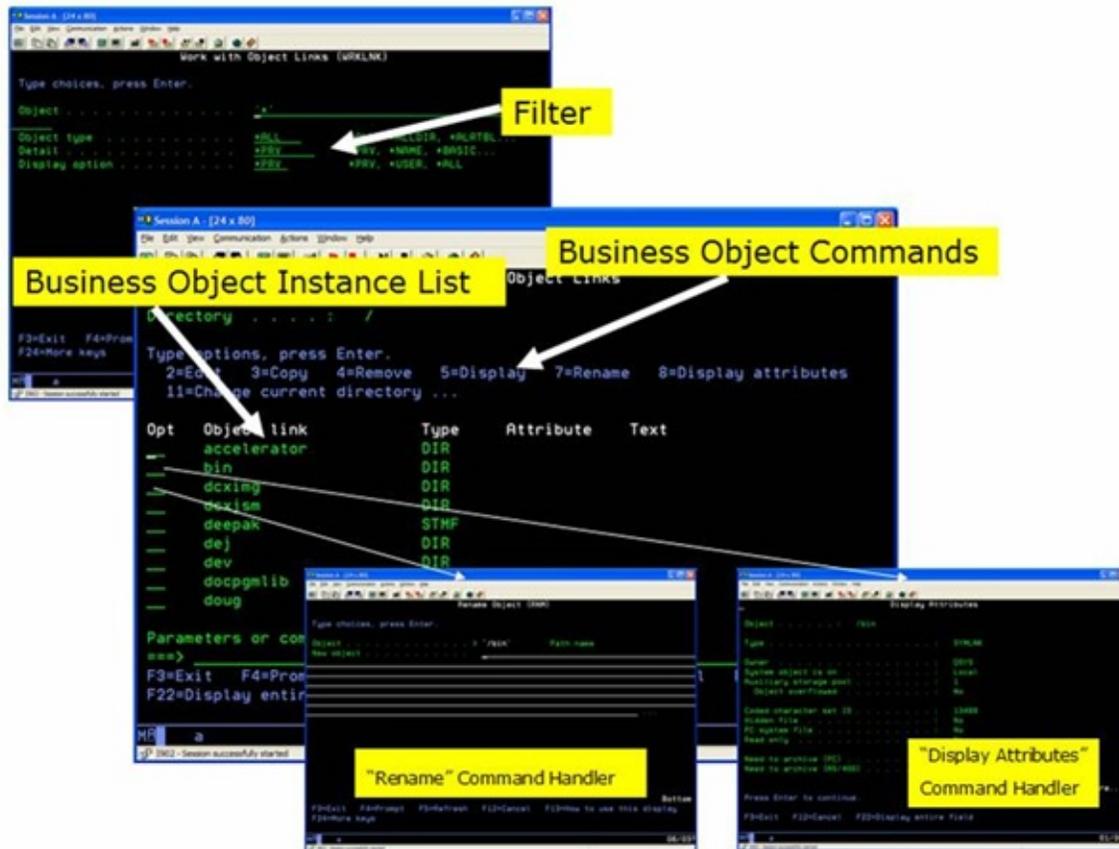
The navigation screens of a System i application are replaced by graphical elements in the Framework, such as filters and instance lists, which the user can use to quickly locate the object they want to work with.

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:



Here you have a:

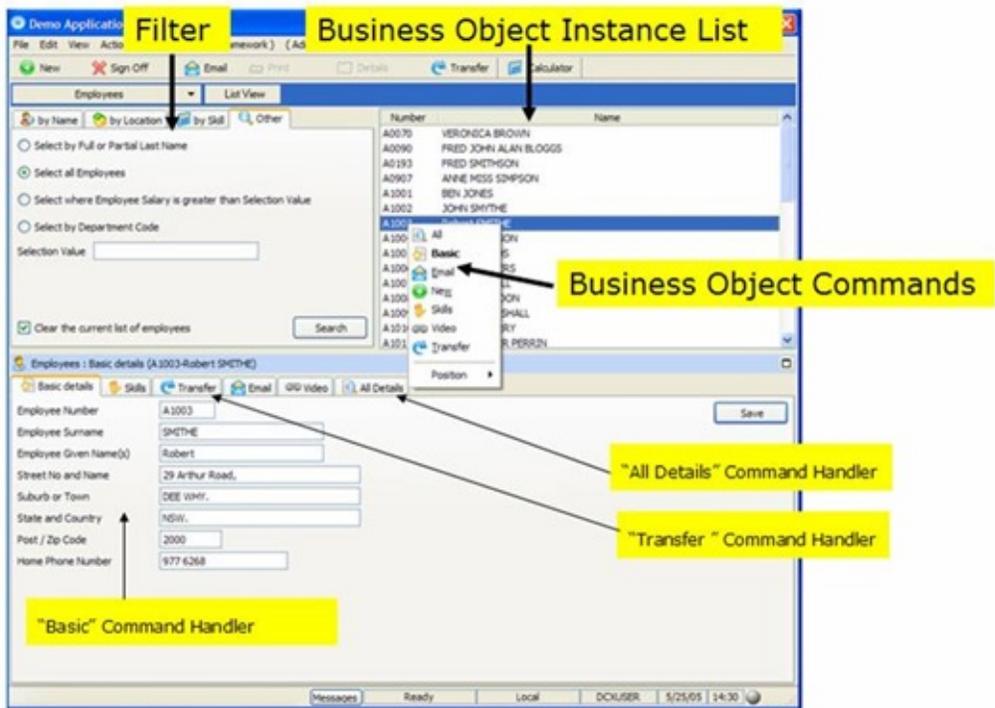
Filter Where the Work with... command provides you with options to filter the list of objects that are displayed. (Many "Work with xxxx" interfaces allow you to filter inside the main display as well).

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?

Reuse, Reface or Rewrite?

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 is 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

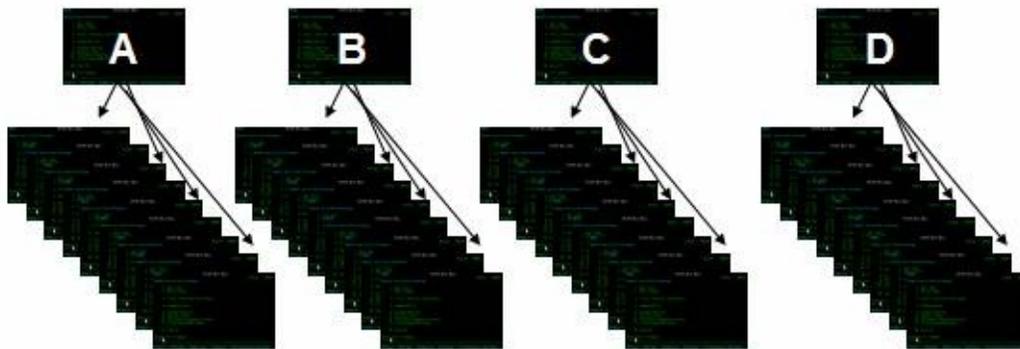
time trying to reuse it and allowing it to become the complete center of attention to the detriment of all the other 5250 programs that also need to be modernized.

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 $4 \times 15 \text{ minutes} = \mathbf{1 \text{ hour}}$.

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.

Reuse, Reface or Rewrite?

In the RAMP context the decisions about Reusing (5250 screens) versus Refacing (using Screen Wrappers) versus Rewriting (new VL component) are complex.

Next time you hear someone say “I could rewrite that in a week”, you need to ensure that they have accounted for:

1. Time to discover and understand all the existing functionality and interfaces.
2. Time to redesign with a modernized UI.
3. Time to code and unit test (this step might actually be a week, as stated).
4. Time to (re)test all the existing functionality and interfaces.

When you hear people say “I can rewrite that in a week” you will often mentally double or triple that to allow for testing, etc. Any x2 or x3 factor you apply might actually need to be x10 or more in rewrite situations (ie: step 3 may be less than 10% of the whole job).

The chum/wrapper solution is useful because it radically reduces the time required to complete steps 1 and 4.

Also, if you are looking at using some sort of template/wizard/code generation techniques, remember that you are only likely to speed up step 3.

ie: If step 3 is only 10% of the job, and you do it twice as fast, you will only have improved the whole project by 5%, not by 50%.

Tutorials for RAMP Windows

A RAMP modernization project should progress like this. You will use an **iterative release cycle**, repeatedly improving your application as time and resources permit:

RAMP Stage 1	↓ Prototype	RAMP-TS001: Creating a Basic Prototype of the Modernized Application RAMP-TS005: Reviewing Design
	↓ Set Standards	Create project standards.
	↓ Decide on Security	Decide on security/authority policy.
	↓ Name screens	RAMP-TS002: Rapidly Modernizing Complete Application Segments RAMP-TS004: Naming and Classifying the Employee Screens
	↓ Classify screens	RAMP-TS002: Rapidly Modernizing Complete Application Segments RAMP-TS004: Naming and Classifying the Employee Screens
RAMP Stage 2A	↓ Rapidly modernize	RAMP-TS002: Rapidly Modernizing Complete Application Segments
RAMP Stage 2B	↓ Individually modernize	RAMP-TS006: Snapping in a Basic Inquiry Screen RAMP-TS007: Snapping in a Data Entry Function RAMP-TS008: Changing Inquiry Screen to Update Mode RAMP-TS009: Tracing

RAMP Stage 2C ↓ **Add value**
See [What Adds Value?](#)

[Navigation](#)

[RAMP-TS015: Understanding and Handling Screen Variations](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS010: Using Special Field Handlers](#)

[RAMP-TS011: Snapping in Shipped Documents Command Handler](#)

[RAMP-TS012: Snapping in Shipped Notes Command Handler](#)

[RAMP-TS013: Sending Instance List Data to Excel](#)

[RAMP-TS014: Snapping RAMP Screens into the HR Demo Application](#)

[RAMP-TSAD04: Redesigning the Screen Using aXes](#)

RAMP Stage 3 ↓ **Remove platform dependencies**

Optionally remove any IBM i platform dependencies.

↓ **Test**

Test the finished application.

↓ **Deploy**

[Deployment Check List for RAMP-TS](#)

← **Repeat**

In every cycle you will move more and more application components from stage 2A to 2B to 2C to 3.

The mix/ratio of stages is critical because it impacts on how long your project will take, and it is dictated by how much time and resource you have available.

Also see [Advanced Tutorials](#).

-
-

Before You Use the Tutorials

Who Should Use the Tutorials?

Tutorials can be used by novice or experienced LANSAs developers who wish to learn how to use RAMP-TS. No advanced Visual LANSAs knowledge is required. LANSAs for the Web training is required if you are using the Framework for Web development.

How Do I Use the Tutorials?

It is recommended that you complete the Tutorials in sequence. Complete the exercises related to the style of application that you are creating. If you are only creating Windows applications, you may wish to skip the WEBEVENT and WAM related exercises.

To allow for more than one developer to use the tutorials, all LANSAs object names will be prefixed with **iii**. You may use any three characters, such as the initials of your name, for the **iii** characters. For example, if your name is John David Smith you can use the characters **JDS**. When asked to create a component named **iiiCOM01**, you will create a component named **JDSCOM01**. Always remember to replace **iii** with your unique 3 characters.

If you are using an unlicensed or trial version of Visual LANSAs, you must use **DEM** to replace **iii**. When asked to create a component named **iiiCOM01**, you will create a component named **DEMCOM01**.

What Partition Should I Use?

You need to use an RDMLX-enabled partition.

It is recommended that you use the DEM partition for the tutorial. The DEM system contains the Personnel System demonstration and all required files used by the tutorial.

If you want to create a WEBEVENT application, do not RDMLX-enable your functions.

Tutorial Installation

In order to use the Tutorials, you must have the Visual LANSAs Framework and RAMP-TS installed. See [Installation and Configuration](#).

The tutorials require the Personnel Demonstration System files (installed by Partition Initialization).

How Many Developers Can Use the Training?

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

Your Feedback

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

RAMP-TS001: Creating a Basic Prototype of the Modernized Application

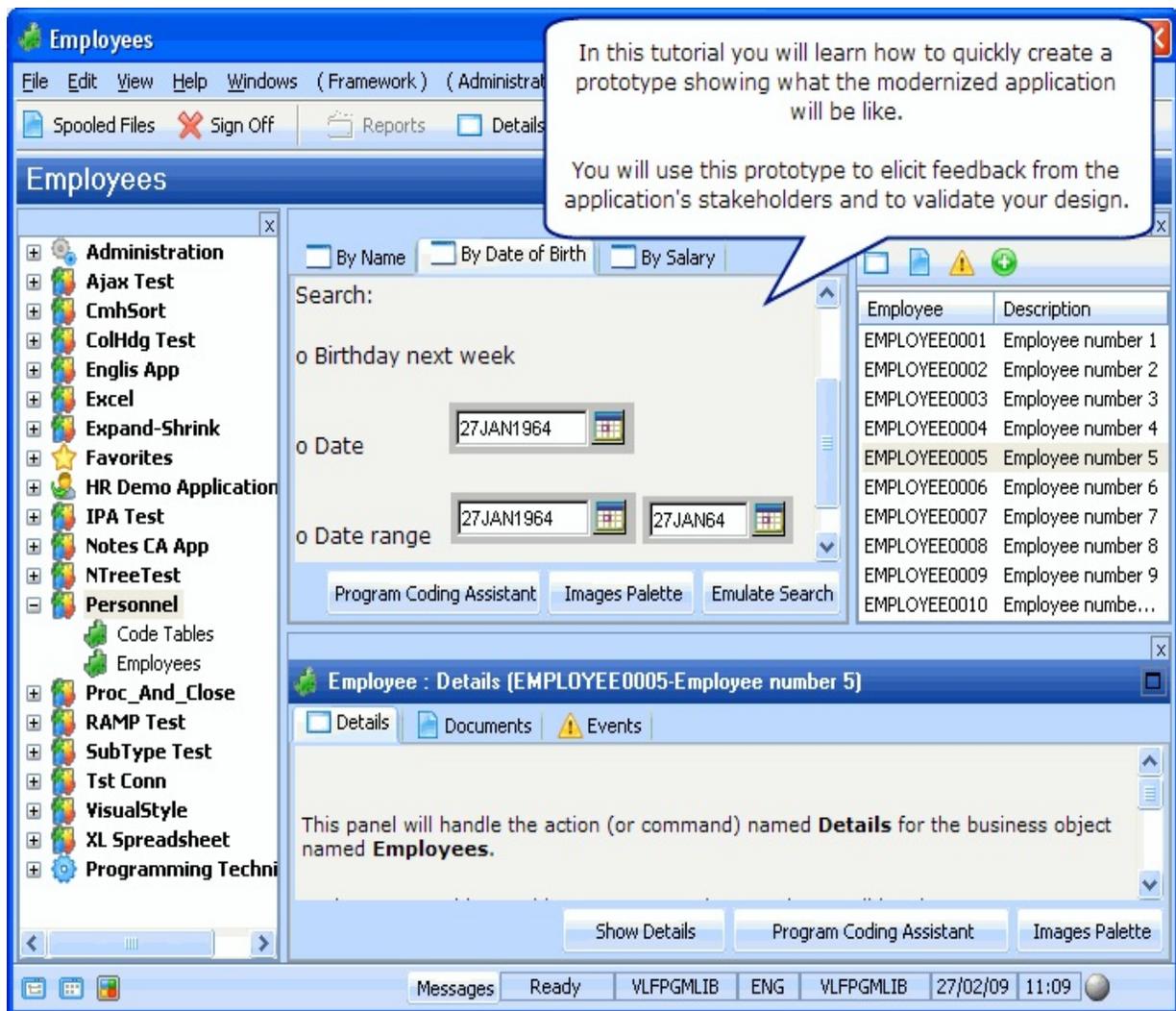
Creating a prototype of the modernized application is the first step you need to perform in using RAMP. A vision of how the completed result will look, act and feel can be formed and executed before any actual modernization steps are taken.

This process acts as a way of rapidly validating your design and uncovering any new or hidden business requirements.

Objectives

To understand how vital this step is in successfully modernizing applications

To learn how to use the Framework prototyping tools



To achieve this objective, you will complete the following steps:

RAMP-TS001 Step 1. Create the Application Prototype

RAMP-TS001 Step 2: Modify the Code Tables Prototype

RAMP-TS001 Step 3: Examine the Employees Prototype

RAMP-TS001 Step 4: Prototype End-User's Access to Employee Information

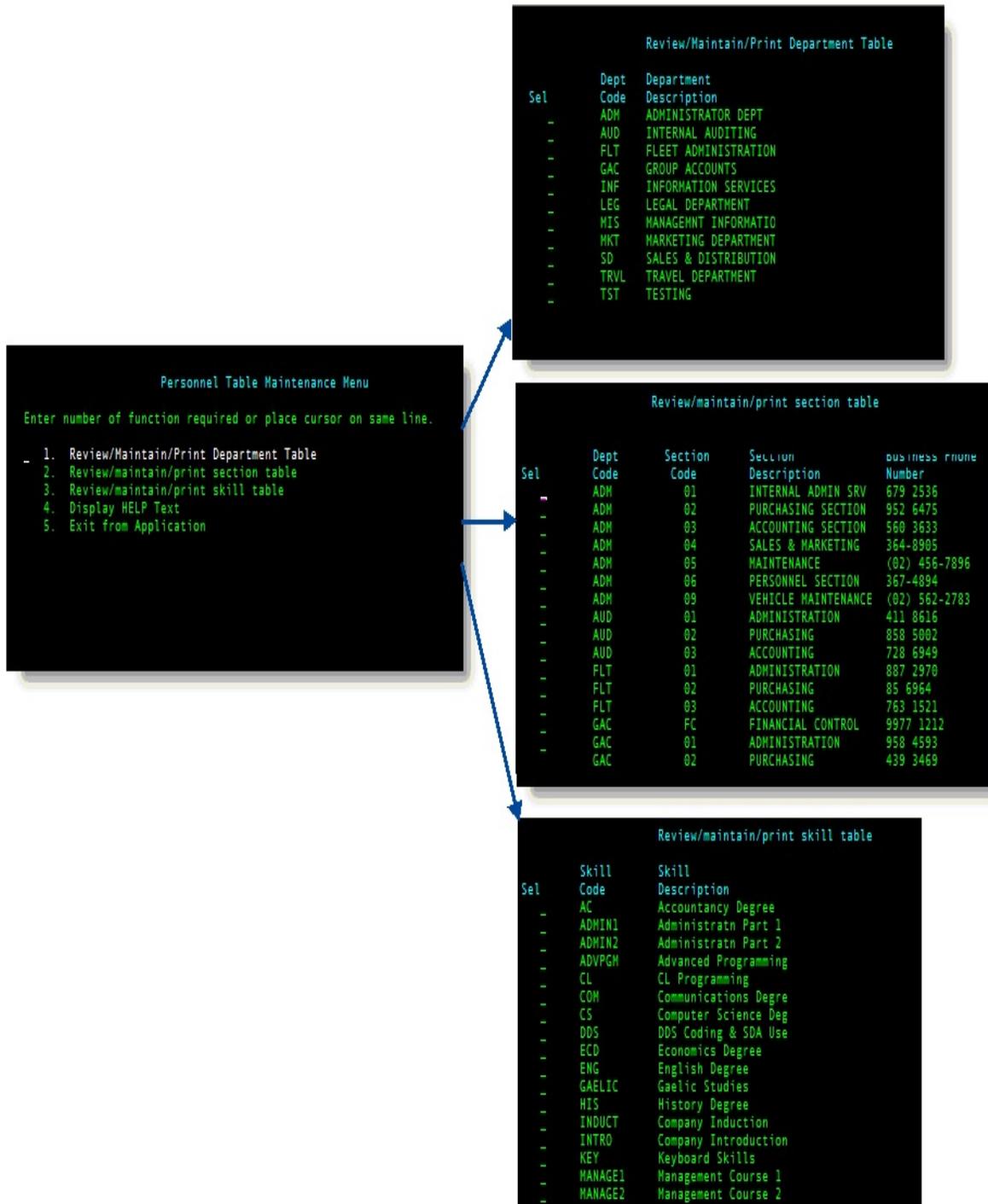
RAMP-TS001 Step 5. Visualize the Filters

RAMP-TS001 Step 6. Validate the Basic Prototype

Application before Modernization

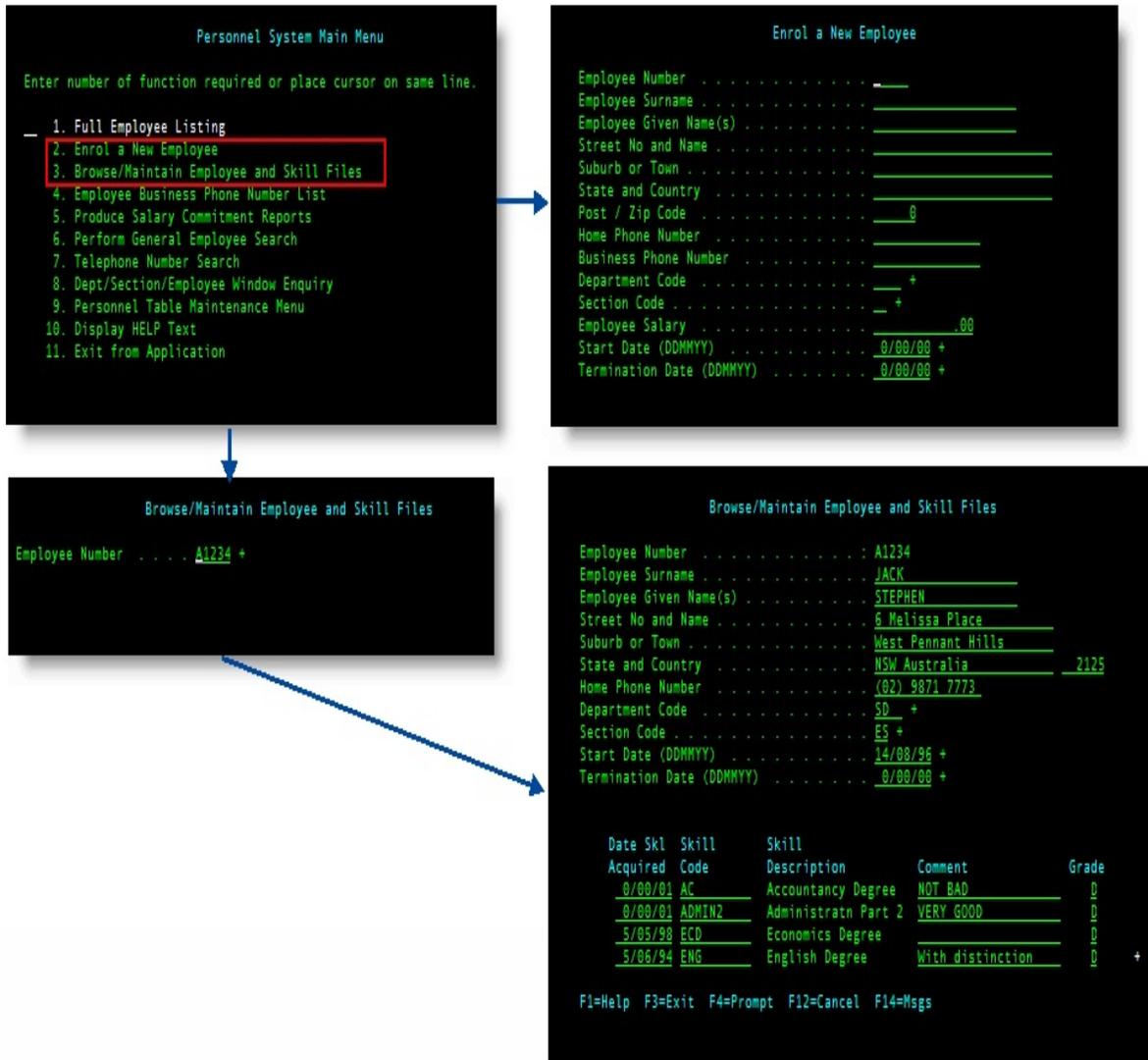
In this tutorial you will be modernizing the Personnel Tables (code tables) and the Personnel System maintenance application.

The Personnel Tables application consists of the Personnel Table Main Menu and various table maintenance functions:



The Personnel Tables application is seldom used, so we plan to modernize it simply by snapping its main menu into the Framework.

The parts of the Personnel System application you will be modernizing are employee skills maintenance and enrolling a new employee:



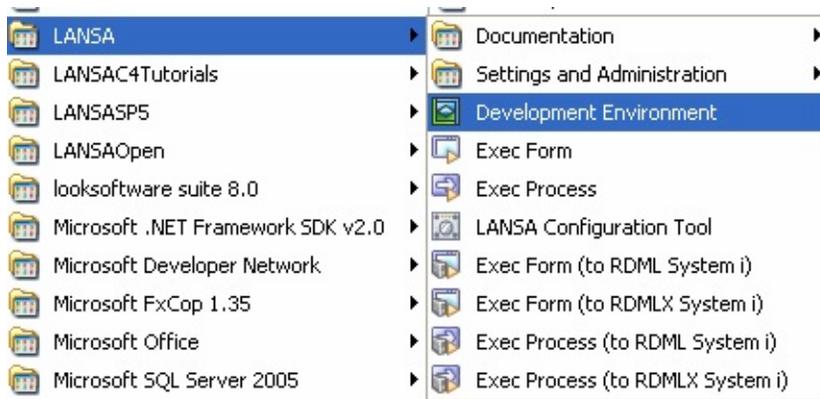
The Personnel System is used frequently, so we will be concentrating the modernization effort in this application.

RAMP-TS001 Step 1. Create the Application Prototype

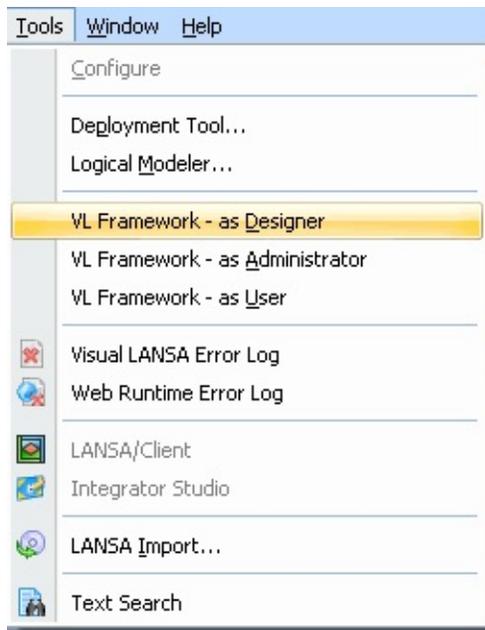
The first step in modernizing your application is always to create a prototype for it. In order to create the prototype you need to identify the business objects the application deals with. To do this see what words the end-users use to describe what the system works with. These words are often reflected in the application menus and screen titles.

In our sample application the users work with Code Tables and Employees, so we will create a Code Tables and an Employees business object.

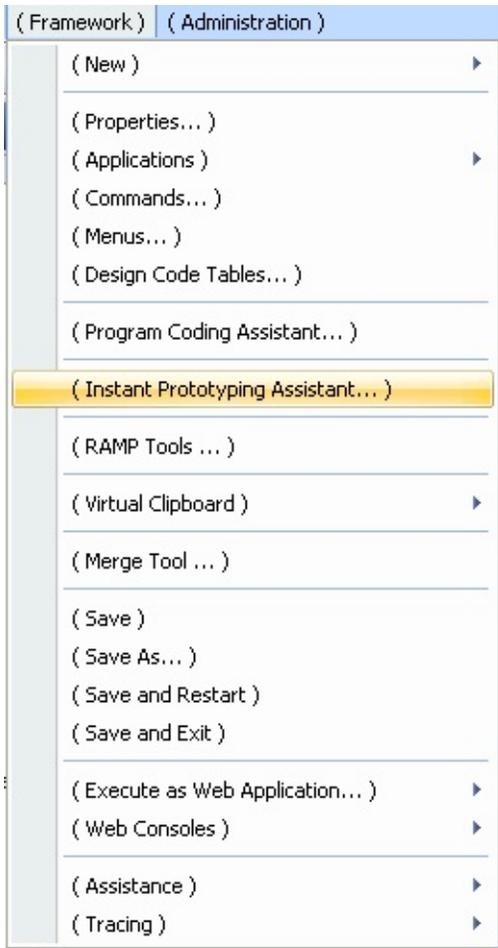
1. Log on to the LANSAC Development Environment:



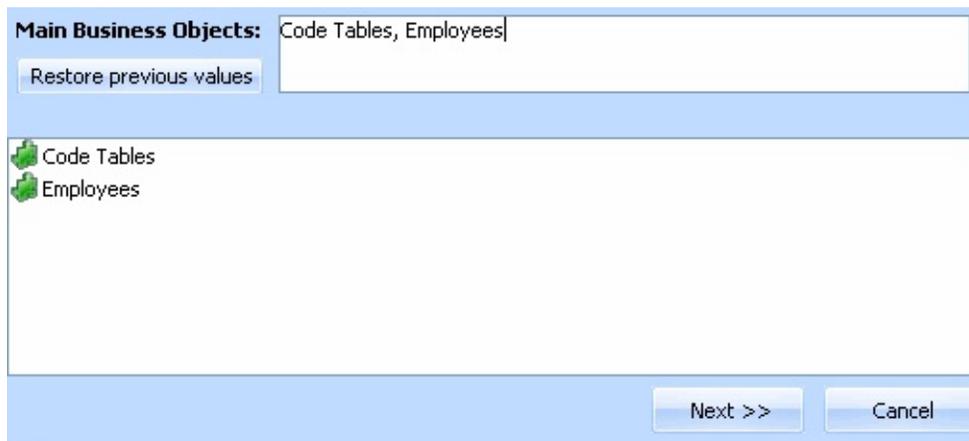
2. In the LANSAC Editor, start the Framework as a Designer:



3. Start the Instant Prototyping Assistant to create the new business objects and application to contain them:



4. Create two new business objects with the name Code Tables and Employees (separate the names with a comma):



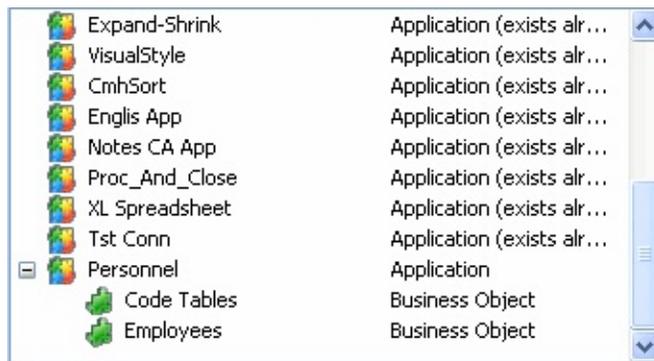
5. Click Next.
6. Associate the Details command with Code Tables by dragging it to the business object:
7. Create two new commands Documents and Events by typing them in the Actions field (separate them with commas):
8. Associate the Details, New, Documents and Events commands with Employees:



7. Click Next.
8. Create a new application called Personnel by typing it in the list of applications:

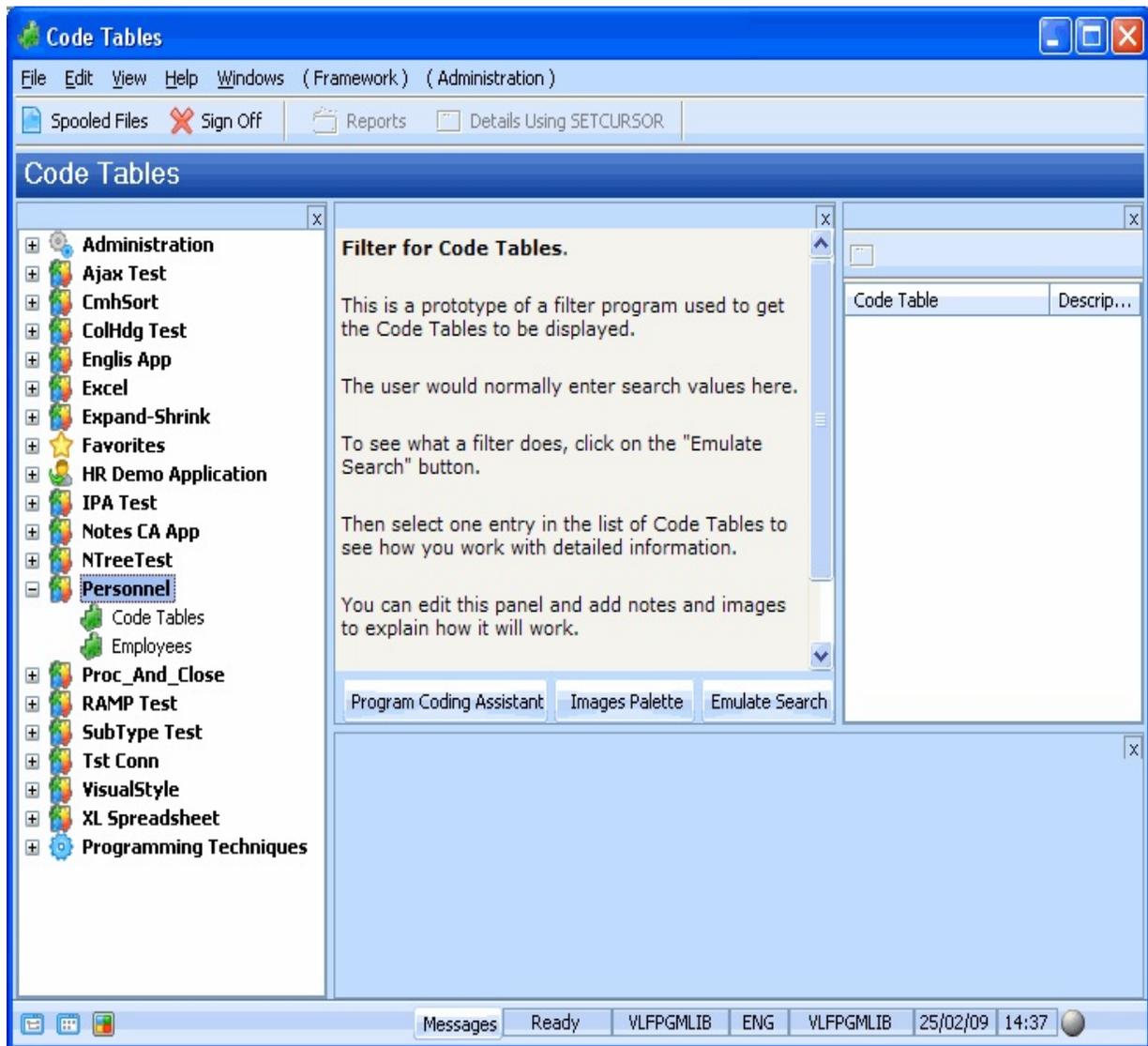
Lastly add Code Tables and Employees to the Personnel application:

9. Drag the Code Tables and Employees business objects and drop them onto the Personnel application :



10. Click Next.
11. Click Finish.

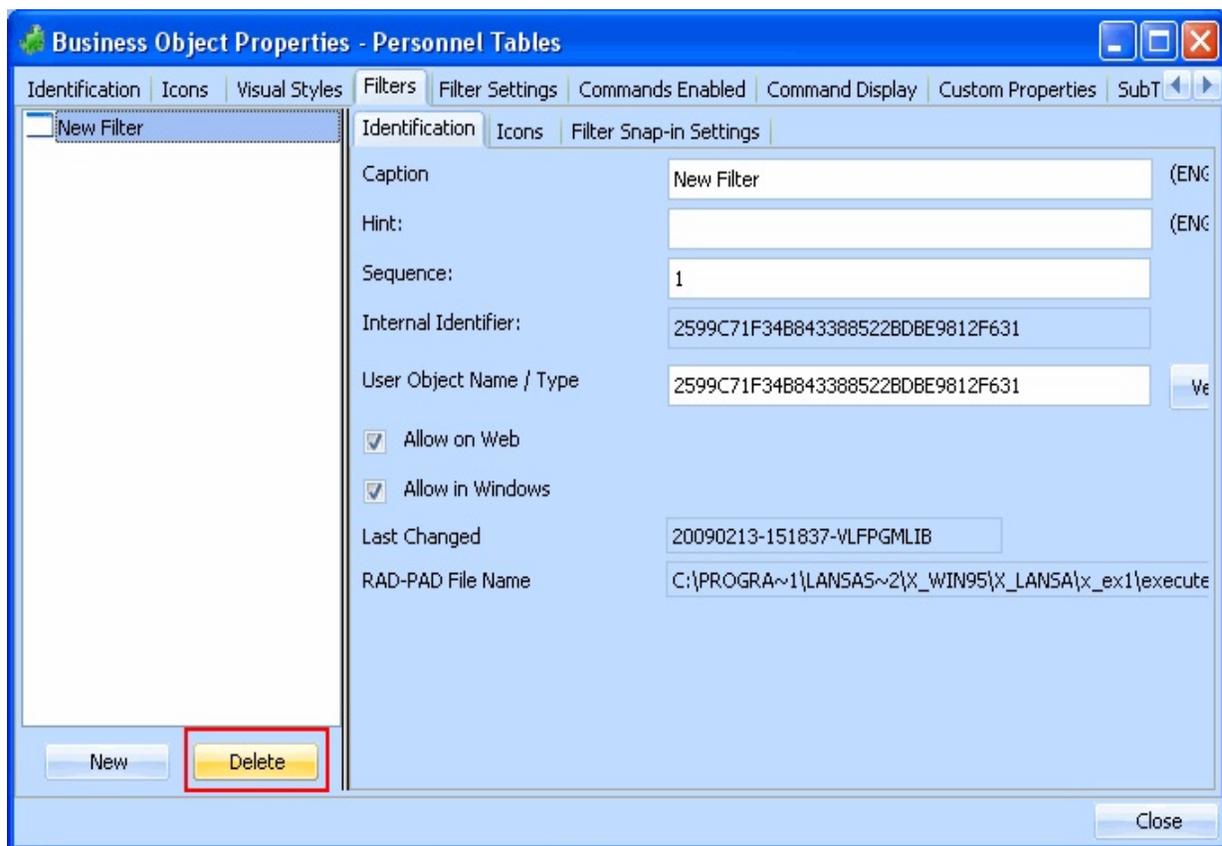
The Personnel application is now created in your Framework:



RAMP-TS001 Step 2: Modify the Code Tables Prototype

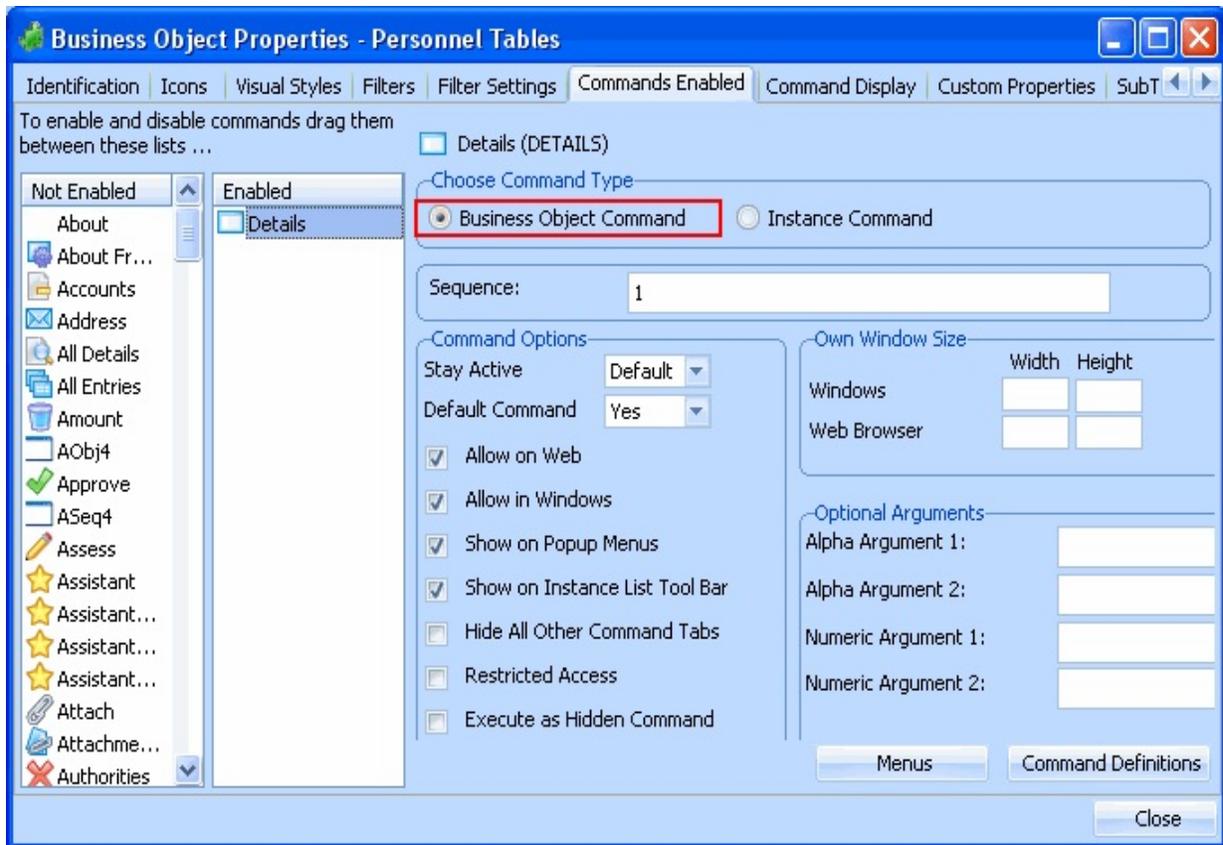
In this step you will modify the properties of the Code Tables business object by removing the default filter created for it. When you are rapidly modernizing an application segment by simply attaching an existing 5250 screen to the Framework, you do not need filters.

1. Double-click Code Tables to display its properties.
2. Display the Filters tab.
3. Delete the New Filter created by default by selecting it and clicking Delete.

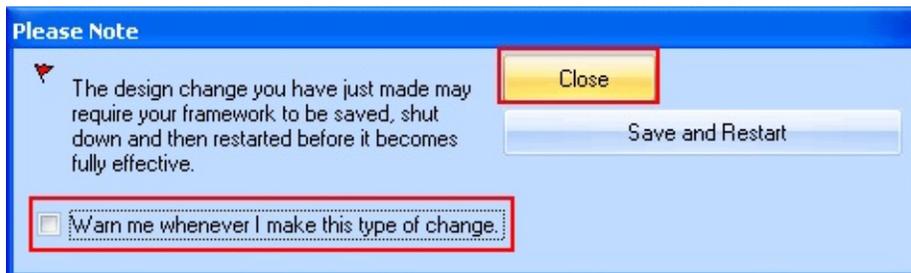


Because there is no filter (and therefore no instance list), you also need to modify the definition of the Details command:

4. Display the Commands Enabled tab.
5. Select the Details command and make its command type Business Object Command.



A message will be displayed asking if you want to restart the Framework.

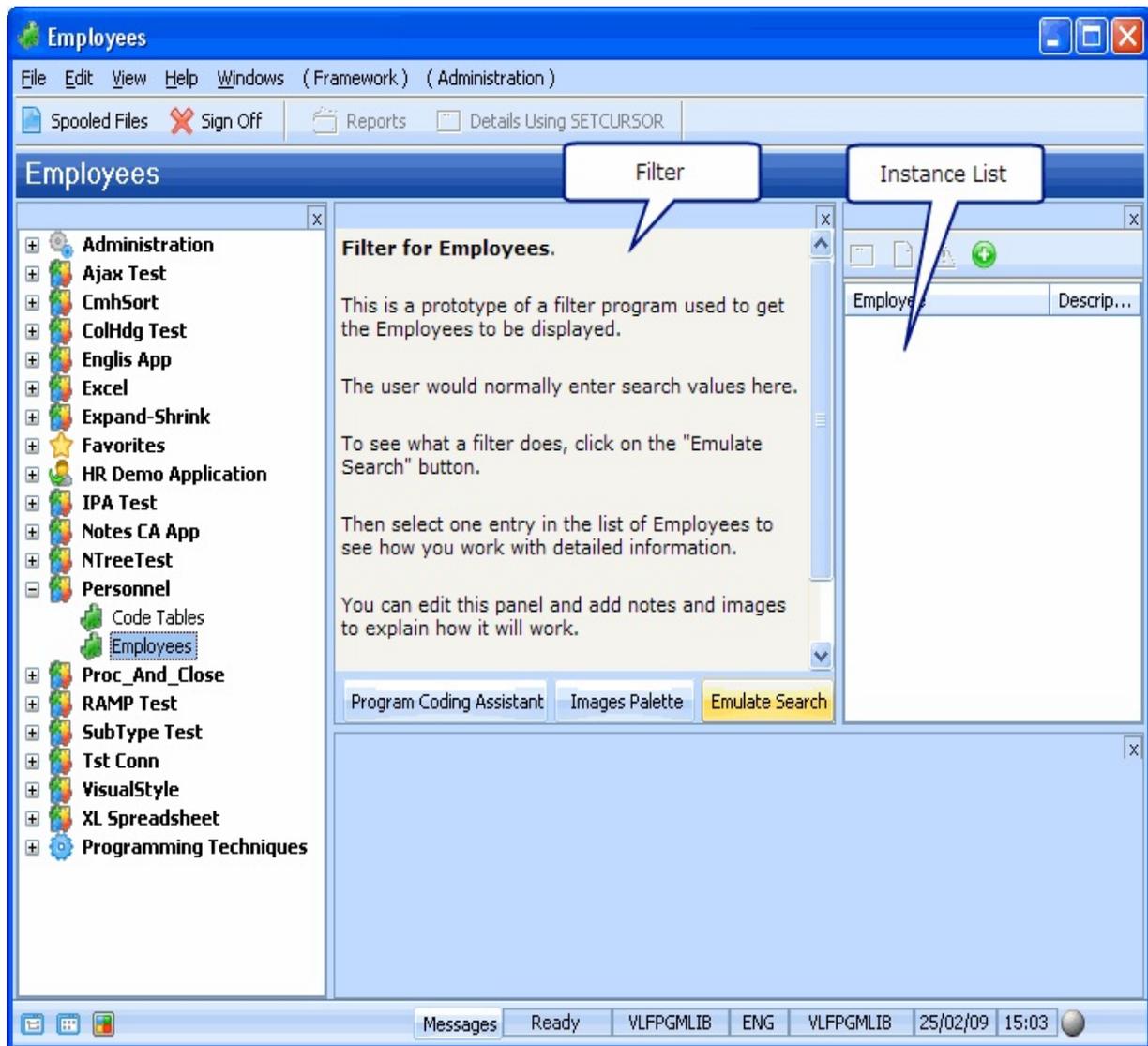


6. Uncheck the message box Warn me whenever I make this type of change and click on the Close button.
7. Close the properties window.
8. Save the Framework.

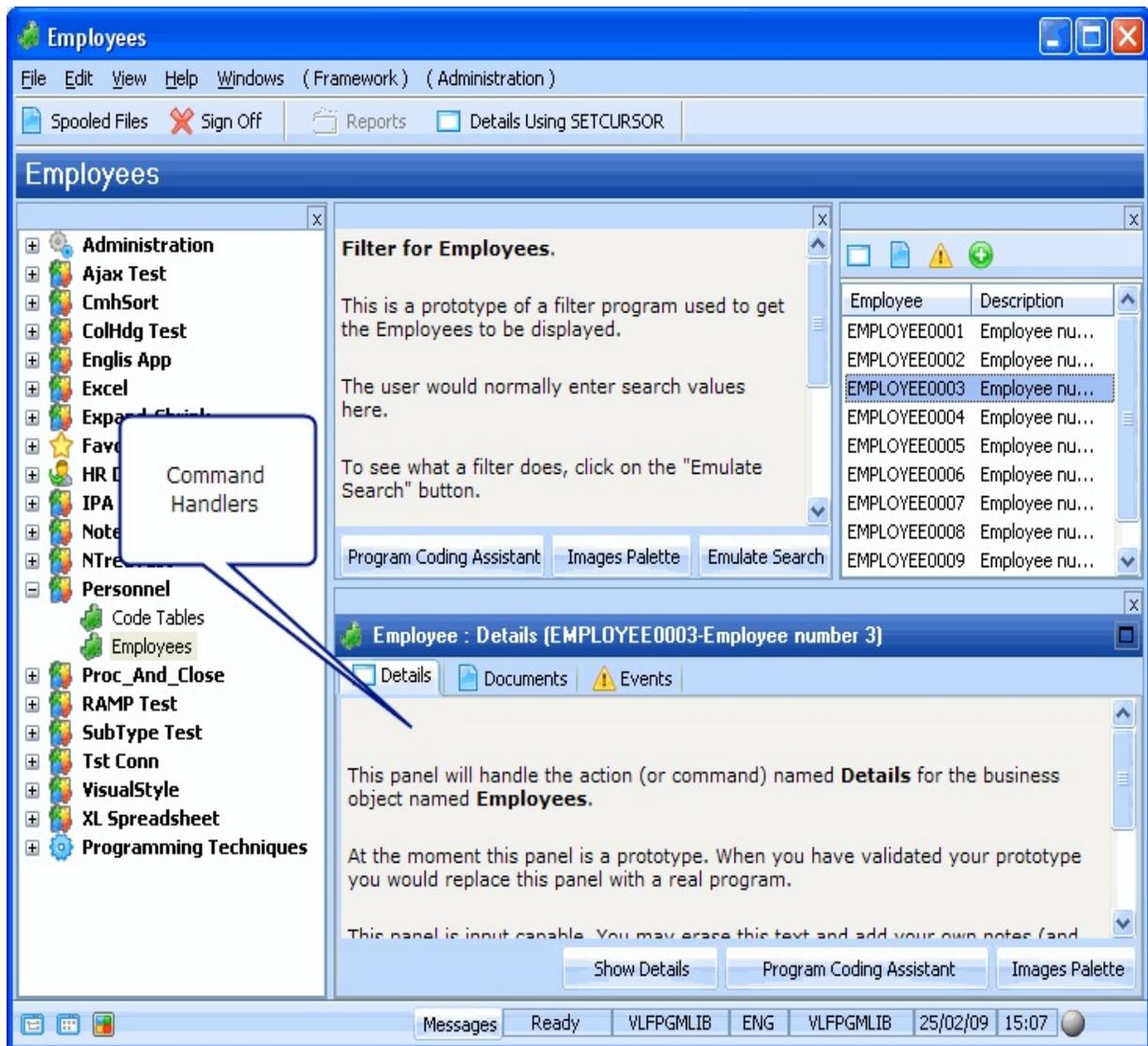
RAMP-TS001 Step 3: Examine the Employees Prototype

In this step you will examine the prototype of the Employees business object.

1. Click on Employees in the Personnel application to display your prototype:

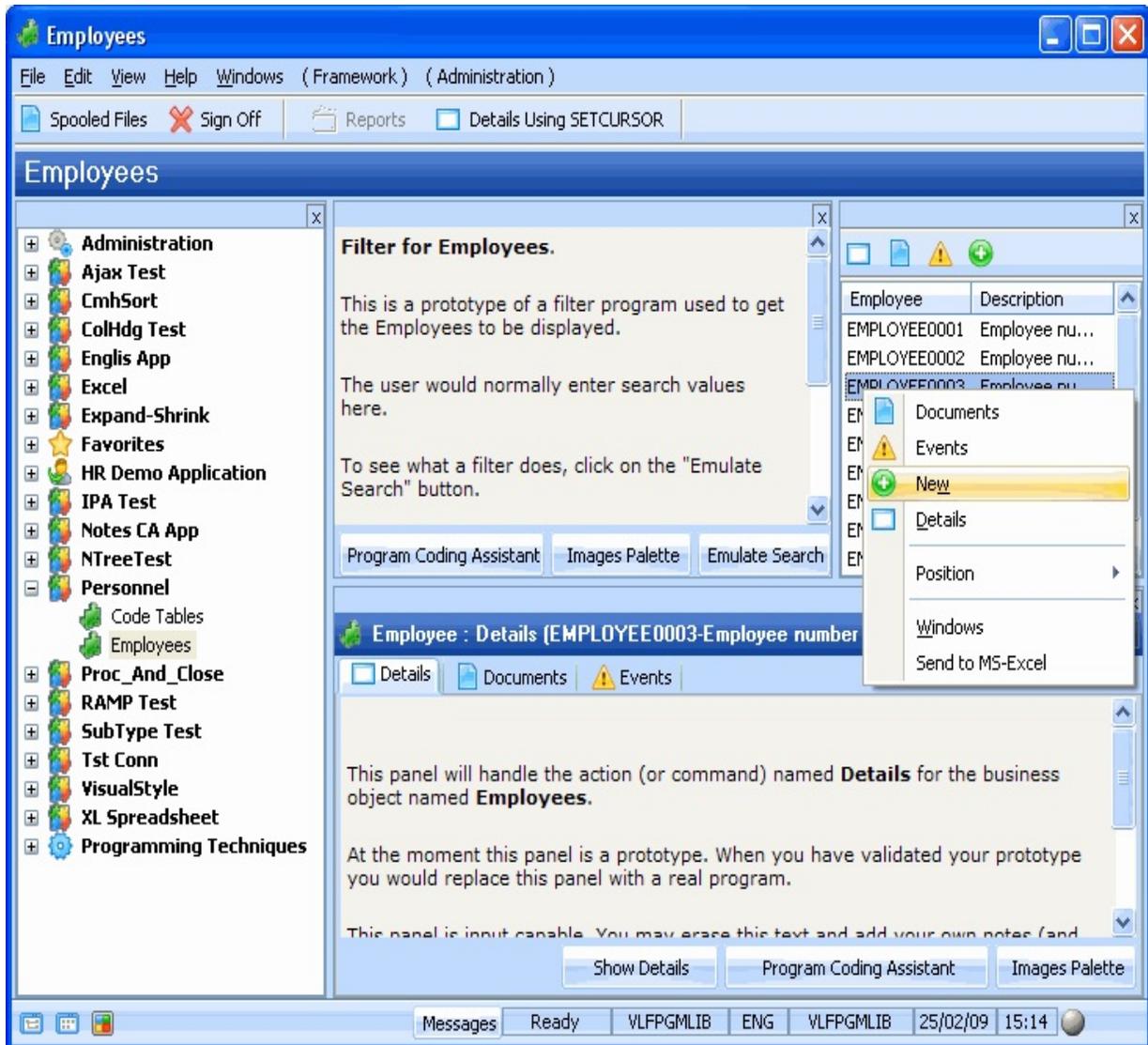


2. Click on the Emulate Search button on your filter to fill the instance list with sample data.
3. Click on one of the sample employees in the instance list to display the command handlers associated with it.



Notice that the command handler for the New command is not displayed with the other command handlers. This is because it is by default set to be a business object command.

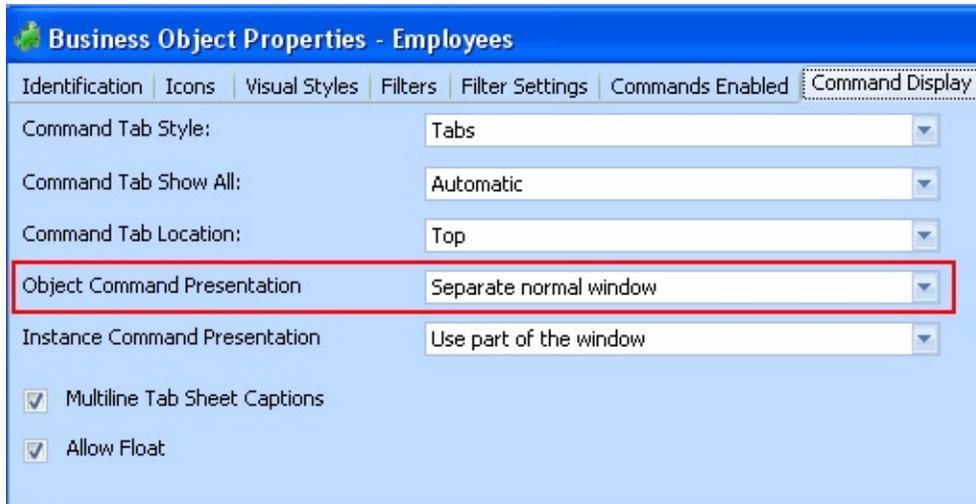
4. Right-click one of the employees in the instance list to display the context menu and select the New command:



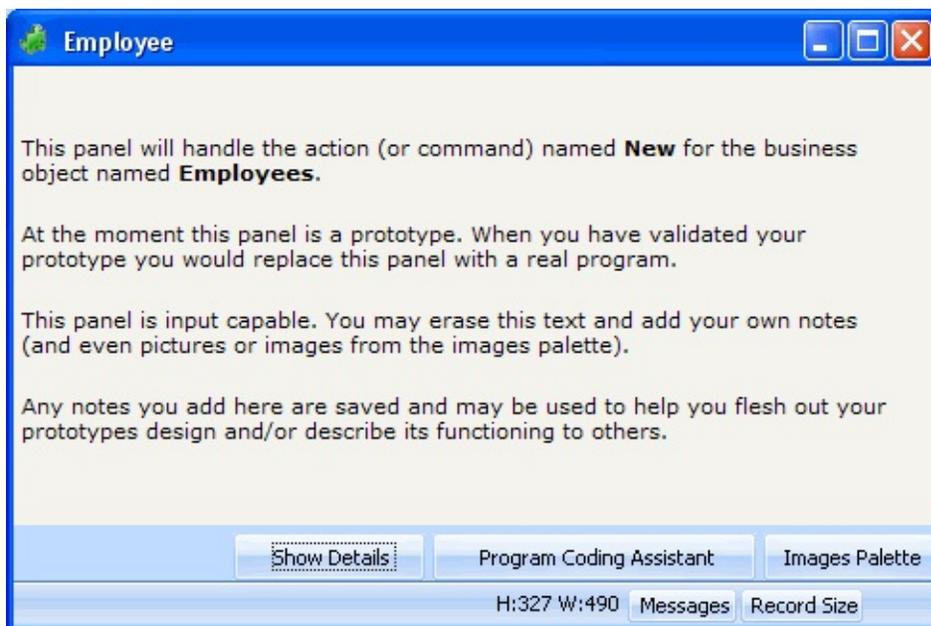
The command handler for New Employee is displayed.

Typically you want business object commands to appear in separate windows unrelated with the instance list, so you will change the definition of the New command so that it is displayed in a separate window.

5. Double-click the Employees business object to display its properties.
6. In the Command Display tab change the Object Command Presentation to Separate normal window:



7. Close the Employee properties.
8. Select the New command. It is now displayed in a separate window:



9. Close the Employee window.
- In the next step you will develop the initial prototype to make it understandable to all stakeholders.

RAMP-TS001 Step 4: Prototype End-User's Access to Employee Information

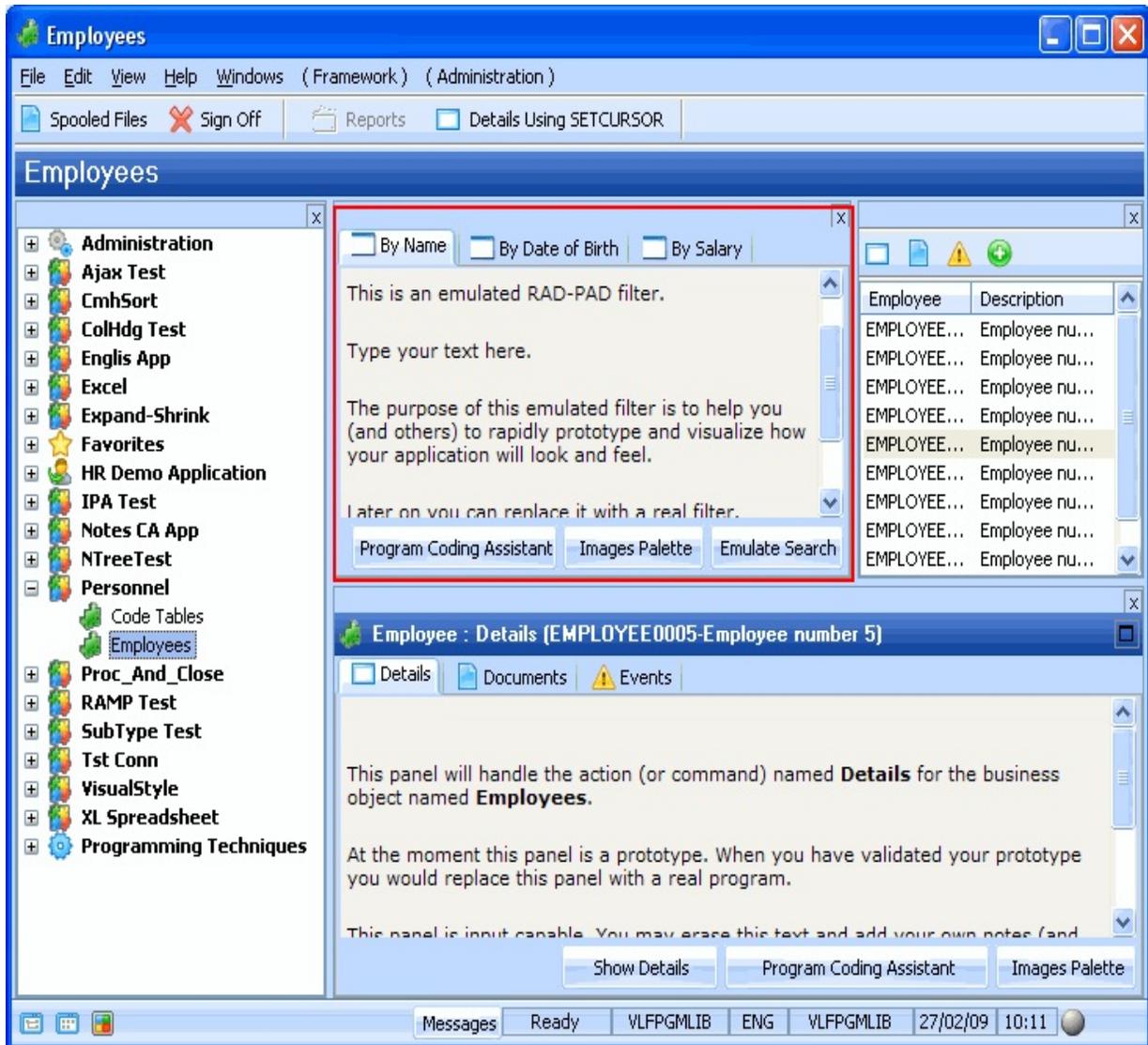
In this step you prototype different ways the employee information can be accessed by creating filters.

You need to provide prototype filters that search the data in various ways that your end users might find useful (regardless of what logical files are defined) in order to elicit their input of how they would like to retrieve employee information.

This step is critical in modernizing your application in a way that will add value to the end users.

1. Double-click the Employees business object to bring up its properties.
2. Display the Filters tab.
3. Use the New button to create two more filters.
4. Make the caption of the first filter By Name.
5. Make the caption of the other filters By Date of Birth and By Salary.
6. Close the Employee properties.

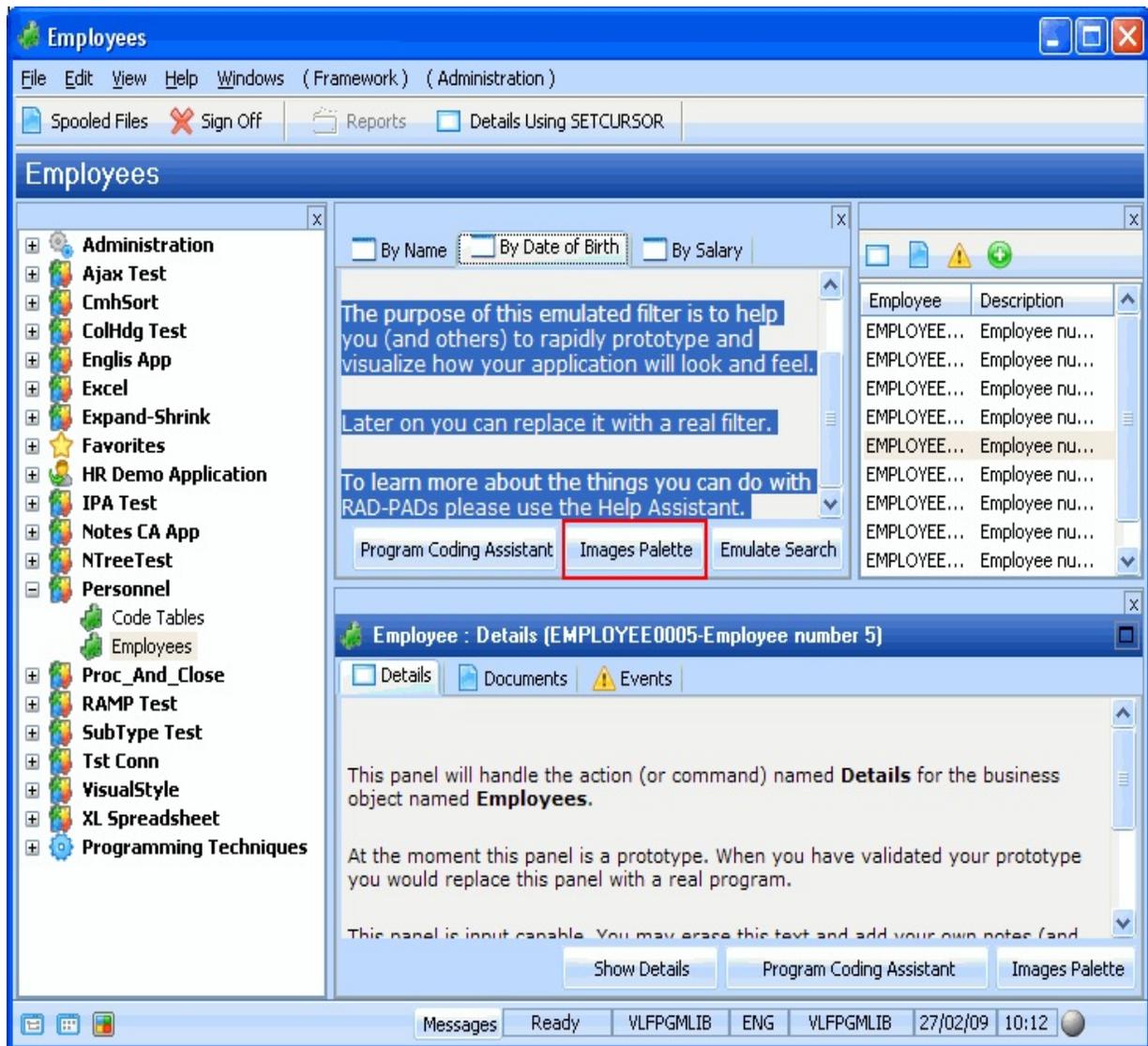
The filters for Employee are now visible:



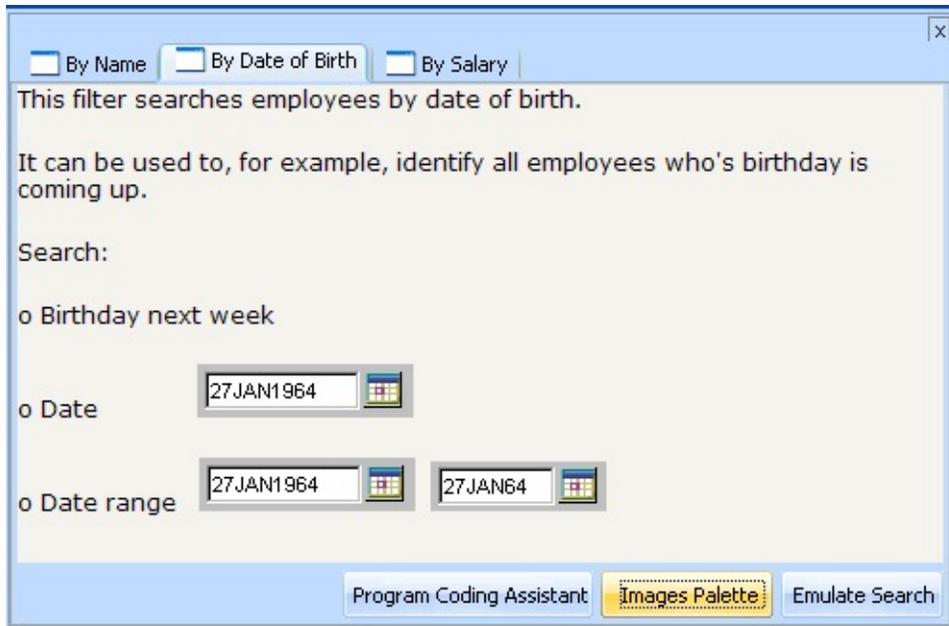
RAMP-TS001 Step 5. Visualize the Filters

In this step you will start to develop the initial prototype to make it understandable to all stakeholders. You will use the RADPADs on the filters to roughly visualize what the filters will look like.

1. Display the By Date of Birth filter.
2. Select all the text in the filter:



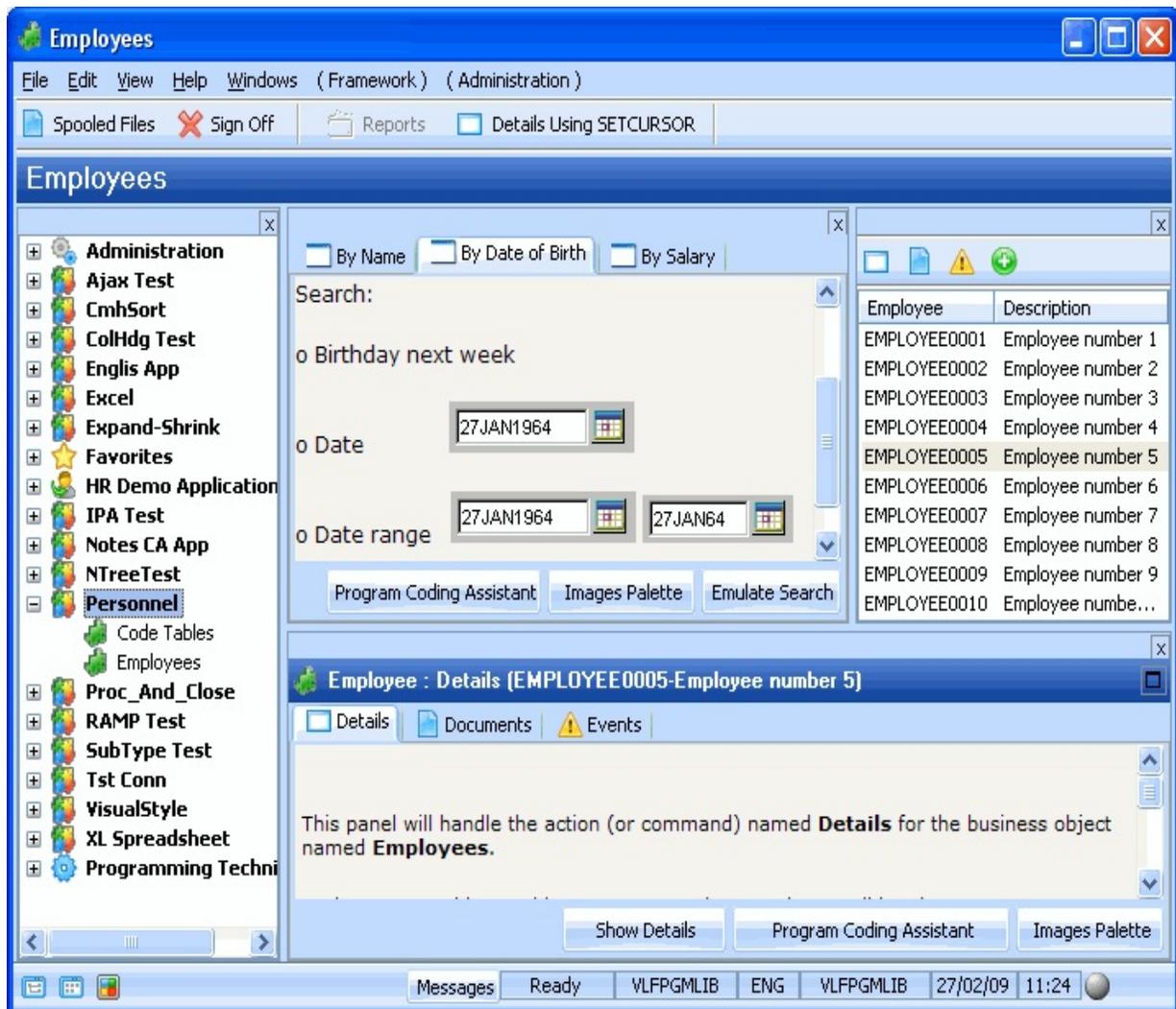
3. Delete the text.
4. Prototype the filter by typing in text inserting pictures using the Images Palette. For example:



5. Quickly prototype the other two filters.
6. Save and restart the Framework.

RAMP-TS001 Step 6. Validate the Basic Prototype

The first stage of prototyping your modernized application is now complete. The prototype shows the basic structure and functionality of the application once it is modernized:



In real life situations at this point you need to show your basic prototype to the stakeholders of the application to ensure that:

Your choice of business objects is understandable and acceptable to them

That the commands you have associated with the business object reflect the

ways they work with the information

That the access to the information that the filters provide is useful and adequate

When prototyping a real application this would be the time to let the end-users try out the prototype. Users typically find it easy to give their input when they have a concrete sample of the system available.

After collecting the feedback, adjust the initial prototype.

The prototype will be refined in step [RAMP-TS005: Reviewing Design](#) .

Summary

Important Observations

You have now completed the basic prototype of the modernized application.

Using the prototype you can:

Validate your design.

Show it to end-users and others to obtain feedback.

Quickly rework your design until it matches all the requirements.

Create alternative solutions.

What You Should Know

How to create an application and a business object in the Framework and how to associate command handlers with the business object.

How to set Framework properties

How to use RADPADs and the Images palette to visualize your prototype.

RAMP-TS002: Rapidly Modernizing Complete Application Segments

You do not have to use RAMP on every single 5250 screen in your application. Often whole application segments are rarely used, and typically these can be snapped into a Framework in a very short time

Understanding when and how to use this approach will impact how long it takes you to complete your RAMP project.

Objectives

To show you how to quickly modernize parts of your application which are not frequently used.

To become familiar with the basic steps in RAMPing an application.

To introduce some key concepts used by RAMP-TS.

In this tutorial you will use RAMP to modernize the menu screen of an application and attach it to the Framework. This single RAMPed screen provides access to the rest of the application.

The purpose of this tutorial is to show how you can very quickly modernize complete application segments. We recommend you do this with all infrequently used parts of an application.

Personnel Table Maintner

Enter number of function required or place c

1.	Review/Maintain/Print Department Table
2.	Review/maintain/print section table
3.	Review/maintain/print skill table
4.	Display HELP Text
5.	Exit from Application

Enter
Msgs
Probe Screen
Documentation
Turn Trace On

Messages Ready VLFPGLIB ENG VLFPGLIB 13/02/09 13:47

To achieve this objective, you will complete the following steps:

[RAMP-TS002 Step 1: Name the Screens](#)

[RAMP-TS002 Step 2: Classify the Screens and Track Navigation](#)

[RAMP-TS002 Step 3: Remove Cancel and Exit buttons](#)

[RAMP-TS002 Step 4: Snap the Application into the Framework](#)

Before You Begin

You need to have your RAMP system set up, for more information see [Installation and Configuration](#).

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

Before You Start

There are some important things you need to bear in mind before you start using RAMP-TS:

Remember Everything is Case Sensitive!

Because RAMP-TS uses Javascript which is case-sensitive, you need to watch your capitalization closely when you name screens and fields, create or call variables, objects and functions.

When you encounter a problem, the first thing you should do is to check the capitalization.

Using a RAMP-TS 5250 Session

A RAMP-Tools 5250 session is heavily tracked and monitored. This means that: A RAMP-TS 5250 session may execute significantly more slowly than an execution (run time) 5250 session.

You should always wait for 5250 design time screens to respond. Avoid typing ahead. Be slow and precise in your screen actions.

Axes developer mode 5250 sessions are designed to perform Axes development work only. This means you should not use Axes developer mode 5250 sessions to do other non-Axes development work (eg: editing your RPG programs).

You should never use Axes developer mode 5250 sessions in end user environments.

Understand How the 5250 Application Works

When RAMPing screens you generate scripts that deal with the way the 5250 application behaves.

Typically 5250 applications exhibit patterns of behavior, because they were created by people who (hopefully) followed design rules and were trying to produce a consistent 5250 UI. So, in a RAMP project:

You will need to discover these application behavioral patterns and accommodate them in your scripts.

The thing to most avoid is discovering “surprising” or “unexpected” behaviors after you have created 200 scripts (say) because this may mean you have to go back and alter them all.

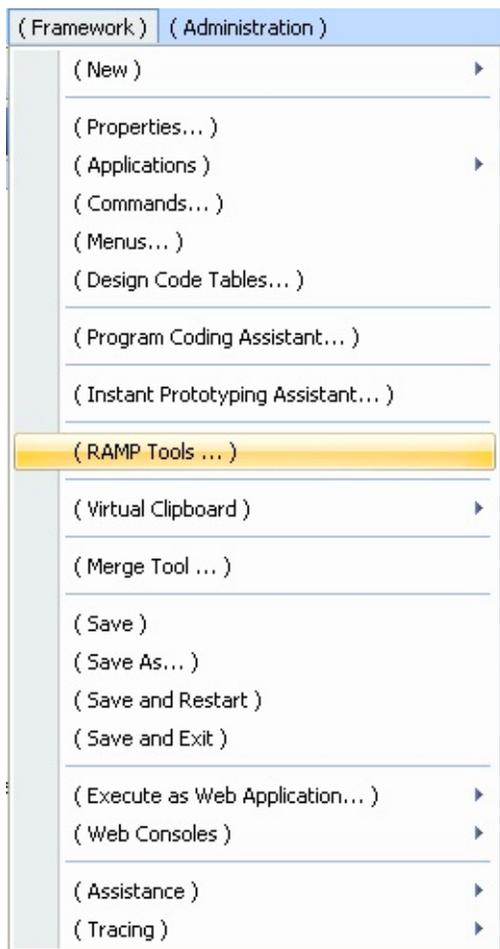
After RAMPing 10 typical 5250 screens (say) you should test them very thoroughly to try to minimize the chance that something surprising or unexpected will be uncovered later in the project.

What you know will not harm your project – the worst it can do is make the project more complicated – it's what you don't know that will – so you need to try to minimize what you don't know.

RAMP-TS002 Step 1: Name the Screens

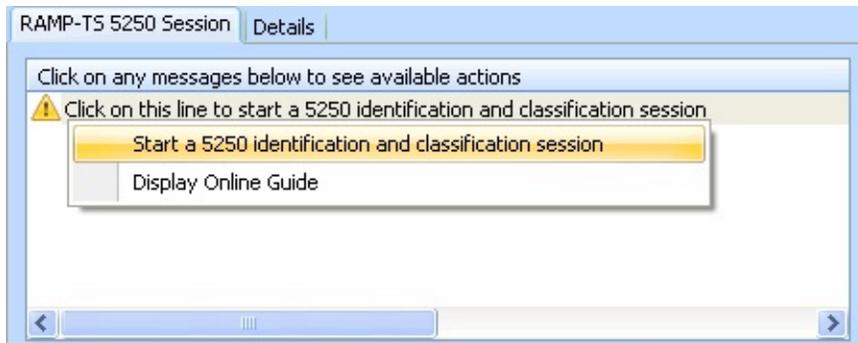
In this step you will name the screens in your application. Naming the screens and the fields used for navigation (if any) is always the first step you need to perform when modernizing applications with RAMP (do not try to track navigation and name screens at the same time).

1. If the Framework is not running, start it.
2. Start the RAMP Tools by selecting the RAMP Tools... option in the Framework menu:



The [RAMP Window](#) is displayed.

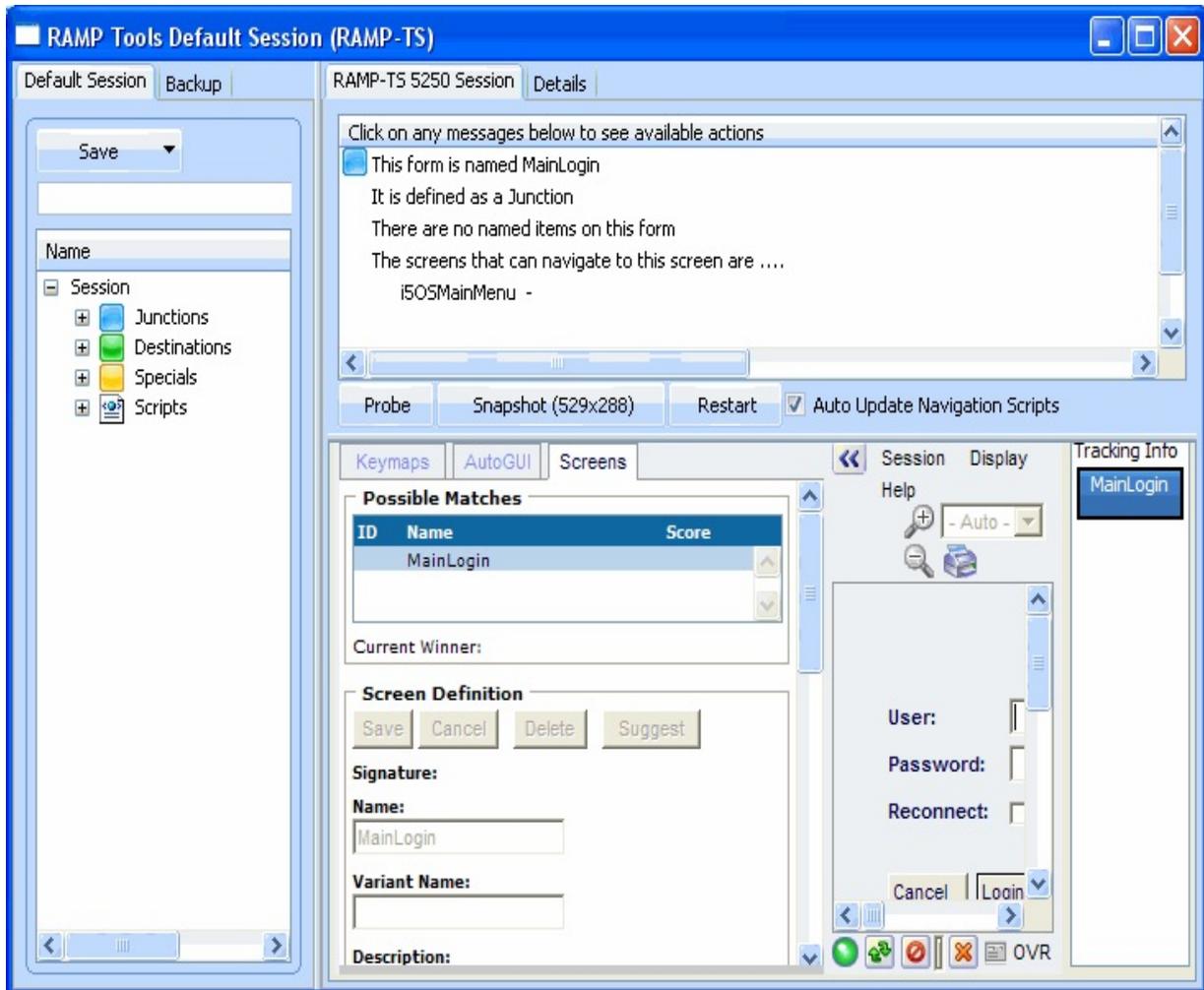
3. In the message area click on the message indicating that RAMP Tools has not been started and choose the option to start it:



4. When the RAMP session starts, connect to your server system (the default RAMP-TS developer authentication userid and password are dev and dev, but your system values may have been changed.)

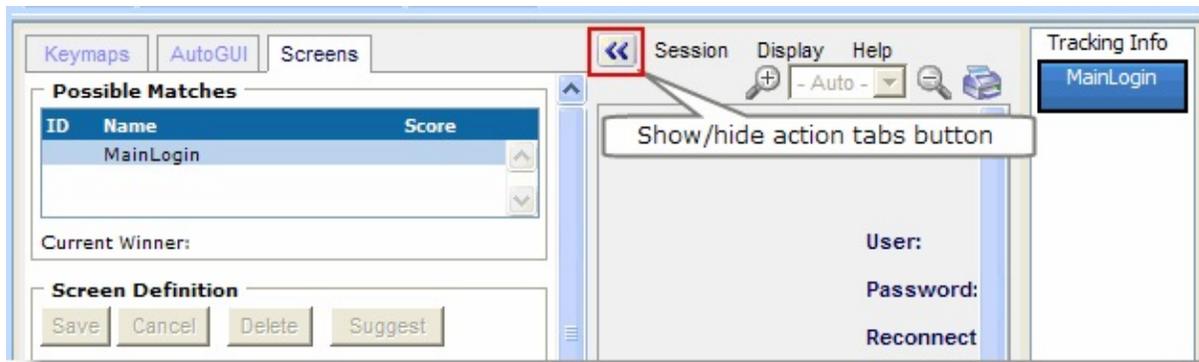


The [RAMP-TS 5250 Emulator Session](#) window is displayed:



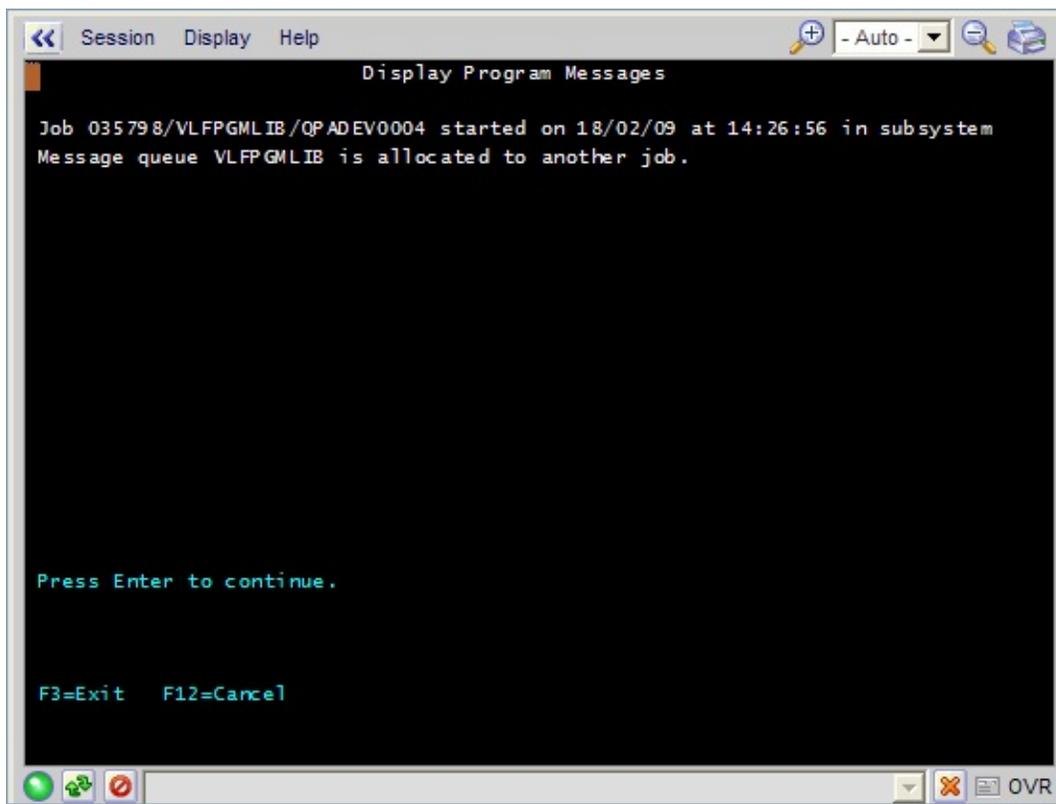
In the Tracking Info area you will notice that the login screen of your server is named by default MainLogin. This is the only screen that has a default RAMP name.

5. Click on the Show/Hide Action tabs button to hide the tabs.



6. Log on to your server system.

The first screen you may encounter is a program message:

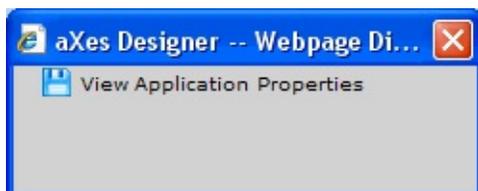


Notice that the [Screen Tracking Area](#) indicates the screen has no name (Unknown Form).

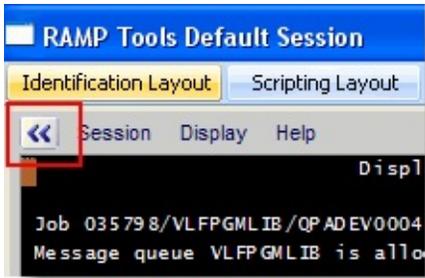
You will now also see another window floating on the left on top of the RAMP Tools window, the aXes Designer:



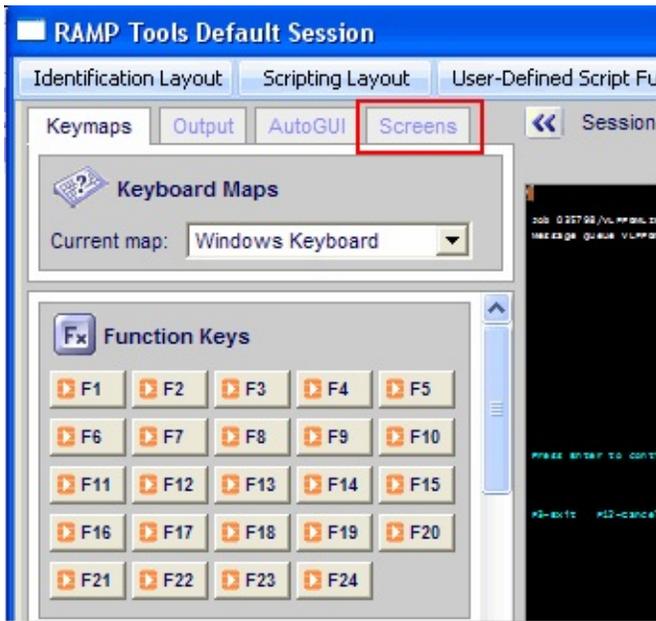
You will use the aXes Designer window in tutorial [RAMP-TSAD04: Redesigning the Screen Using aXes](#) . For now, just size it as small as you can (do this in the subsequent tutorials as well):



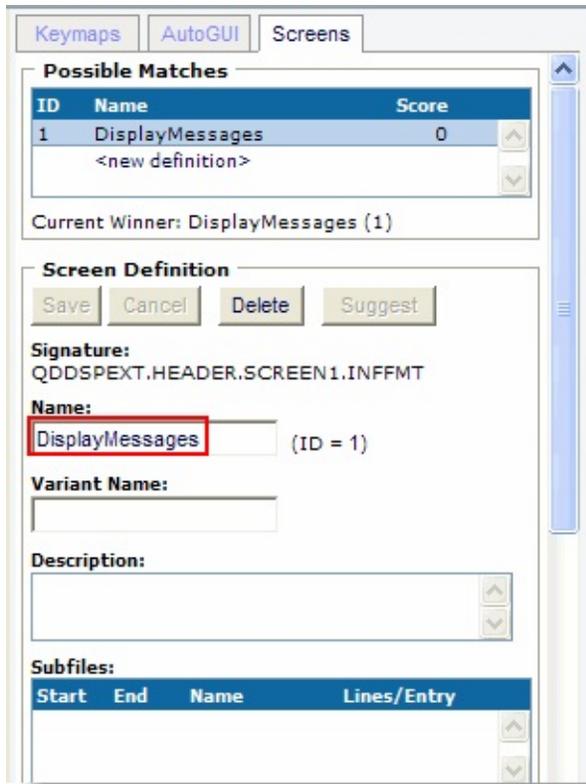
6. To name the screen, click on the Show/hide action tabs button the top-left corner of the RAMP-TS 5250 window:



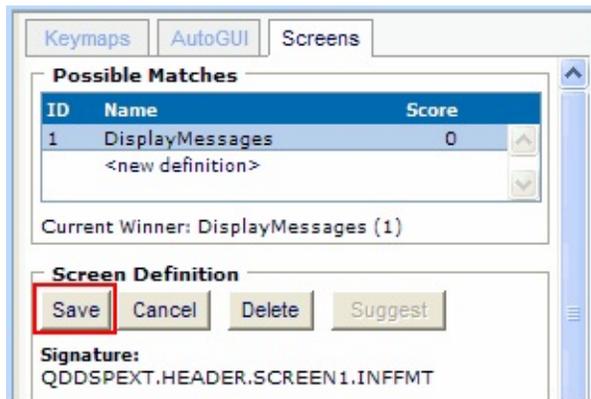
7. Select the Screens tab in the tab folder that appears:



8. Name the screen DisplayMessages:



9. Click the Save button:



The tracking information now shows the name of the screen:



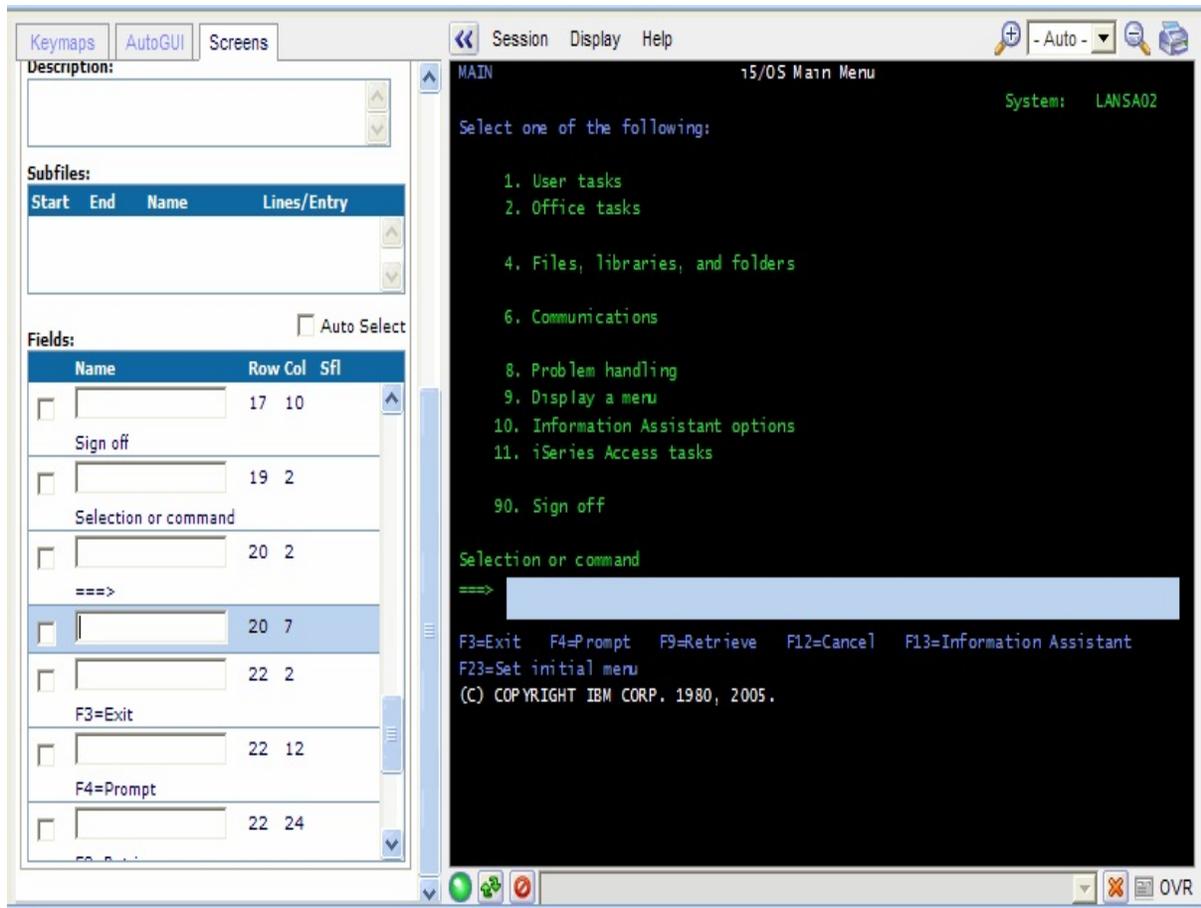
10. Move the focus to the Session window and press Enter to dismiss the message. The i5/OS Main Menu is displayed. The tracking information indicates that the screen does not have a name.

11. Name it i5OSMainMenu in the same way as you named the messages screen.

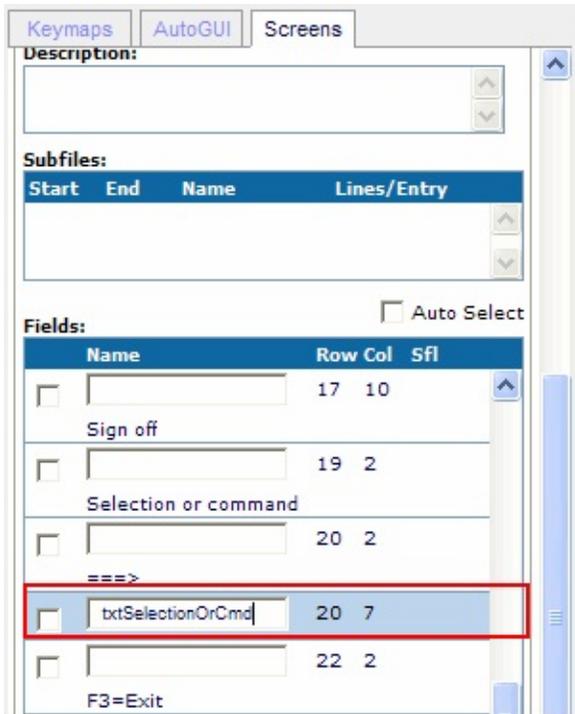
You will also need to name the command line field on this screen so that any commands entered can be tracked:

12. Scroll down the list of fields in the Screen tab.

13. Locate the command line field:



14. Name the field txtSelectionOrCmd and click Save.



15. Display the Personnel Table Main Menu by typing this command on the command line on the i5/O5 Main Menu screen:

```
lansa run psltab partition(dem)
```

16. Identify the screen with the name PSLTABMain.
 17. Click on the arrow button to hide the tab folder.
- You have now finished naming your screens.

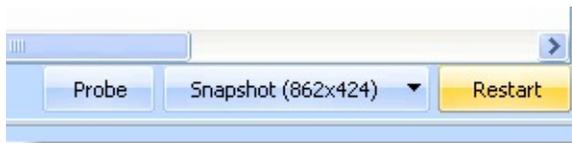
RAMP-TS002 Step 2: Classify the Screens and Track Navigation

In this step you will classify the screens and track the navigation in your application. When you classify a screen a script that records its navigation route is created for it.

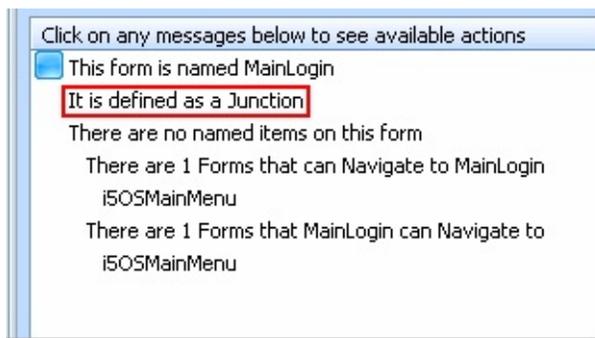
Before this step you might want to read this FAQ: [I have made a mistake in classifying a screen. How do I change the classification?](#)

When navigating 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).

1. Exit from the Personnel Table Main Menu.
2. Sign off.
3. Restart navigation tracking by clicking on the Restart button under the Message Area:



Notice that the Message Area indicates that the login screen has already been classified as a junction:

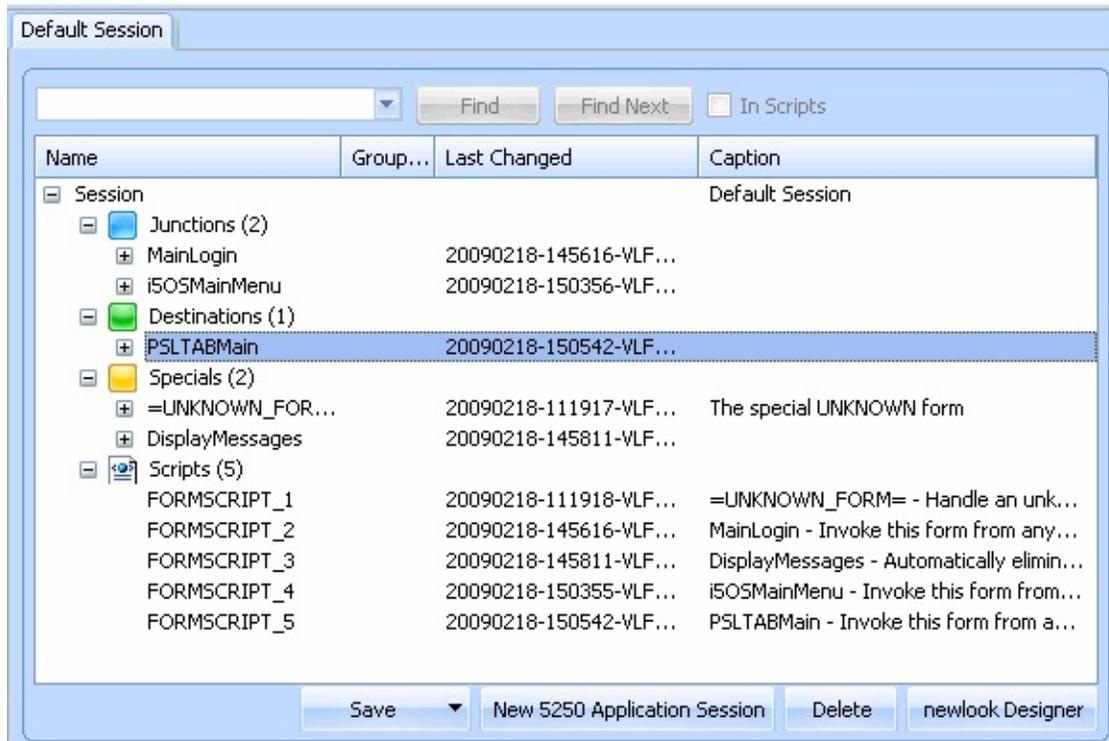


4. Log on to your system.
5. Classify the message screen as a Special form. Depending on your system this may take a while, please wait until RAMP has assigned the screen type.

RAMP-TS002 Step 3: Remove Cancel and Exit buttons

In this step you will remove the Cancel and Exit buttons from the Personnel Table Main Menu screen so that users cannot navigate to other parts of the application from this screen.

1. Display the Default Session tab.
2. Select PSLTABMain in the list of Destination screens:



The PSLTABMain screen details are shown on the right (you may have to rearrange the RAMP window to see all the details).

3. In the PSLTABMain Function Key Enablement list, deselect the Exit and Cancel keys:

PSLTABMain

Caption

Grouping

Default newlook Layout Dimensions

Height Width Top Left

Top Mask Height Bottom Mask Height

RAMP Screen Layout Style

Session Fixed Layout

Targets

os400main

Save

Targeted By

os400main

Save

Function Key Enablement

Key	Caption	Enable S	Enable	Seq
F3	Exit	<input type="checkbox"/>	<input type="checkbox"/>	4
F4	F4	<input type="checkbox"/>	<input type="checkbox"/>	5
F5	F5	<input type="checkbox"/>	<input type="checkbox"/>	6
F6	F6	<input type="checkbox"/>	<input type="checkbox"/>	7
F7	F7	<input type="checkbox"/>	<input type="checkbox"/>	8
F8	F8	<input type="checkbox"/>	<input type="checkbox"/>	9
F9	F9	<input type="checkbox"/>	<input type="checkbox"/>	10
F10	F10	<input type="checkbox"/>	<input type="checkbox"/>	11
F11	F11	<input type="checkbox"/>	<input type="checkbox"/>	12
F12	Cancel	<input type="checkbox"/>	<input type="checkbox"/>	13
F13	F13	<input type="checkbox"/>	<input type="checkbox"/>	14

Associated Command

Link to Command (

New

Notes

Details

New

Notes

Details

Session Id *AL

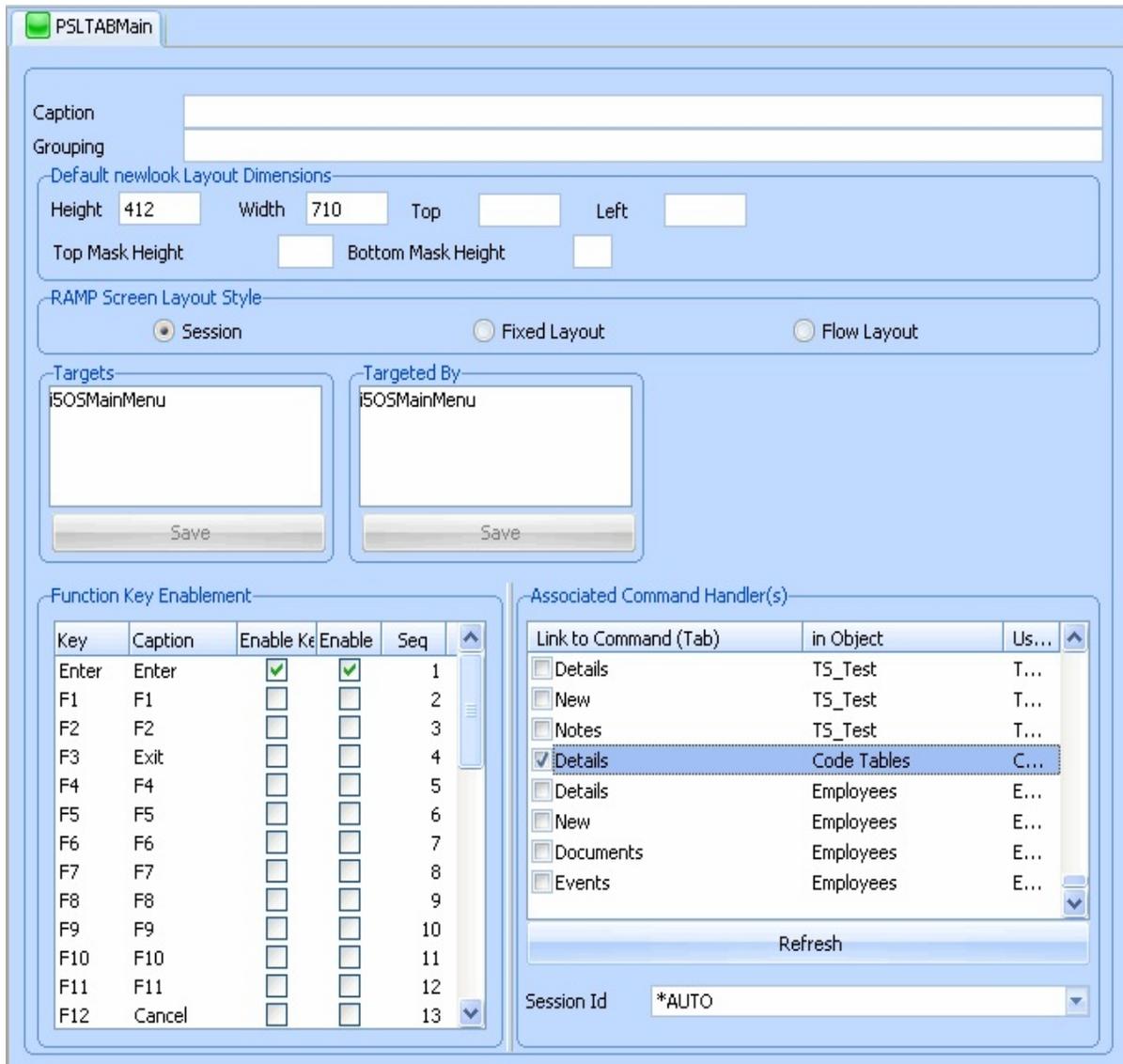
RAMP-TS002 Step 4: Snap the Application into the Framework

In this step you will snap your application into the Framework. You will also specify that an unmanaged session is to be started for the PSLTABMain screen.

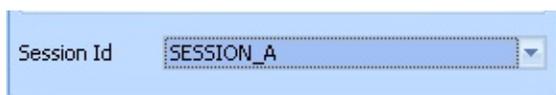
Unmanaged sessions are primarily used to log the user on and then drive them to a specific starting point. From that point forward the user can move around inside the 5250 application without being managed by the Framework.

So, when the user clicks on Code Tables in the Framework menu, the 5250 session logs the user on and then drives them to the menu screen. The entire Code Tables facility (which could be composed of hundreds of 5250 screens) is now accessible without the need to name, classify and track the screens. If the user goes away from the command handler and then comes back again later, the current 5250 screen, whatever it is, is just redisplayed.

1. In the PSLTABMain screen details scroll the list of available commands in the Associated Command Handler(s) list and locate the Details command of the Code Tables business object (note that you can sort the columns in the list when looking for the command). Select the command:



2. In the PSLTABMain screen details, set the Session Id to Session A:

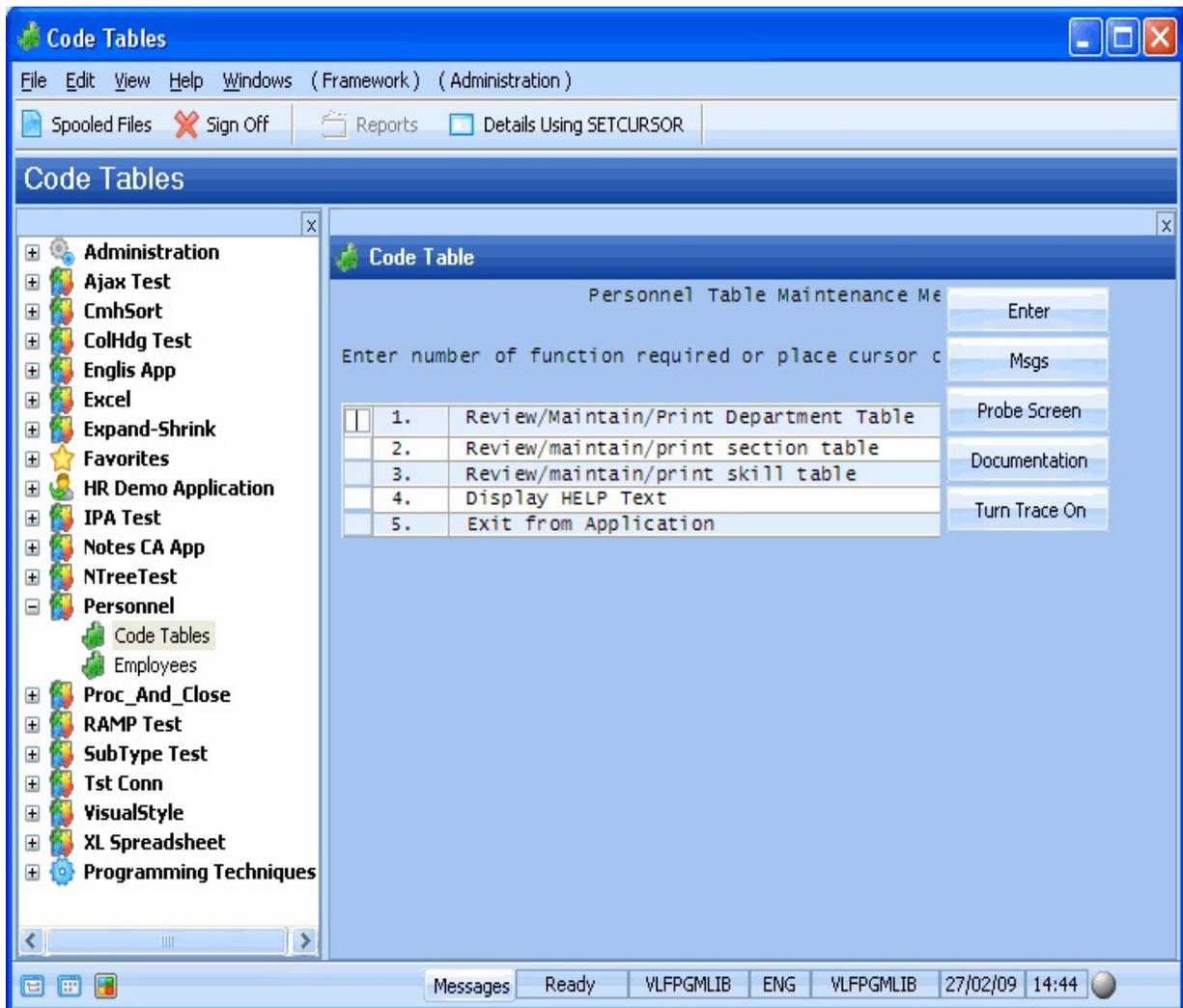


3. Save your RAMP definitions by clicking the Save button and select Partial Save:

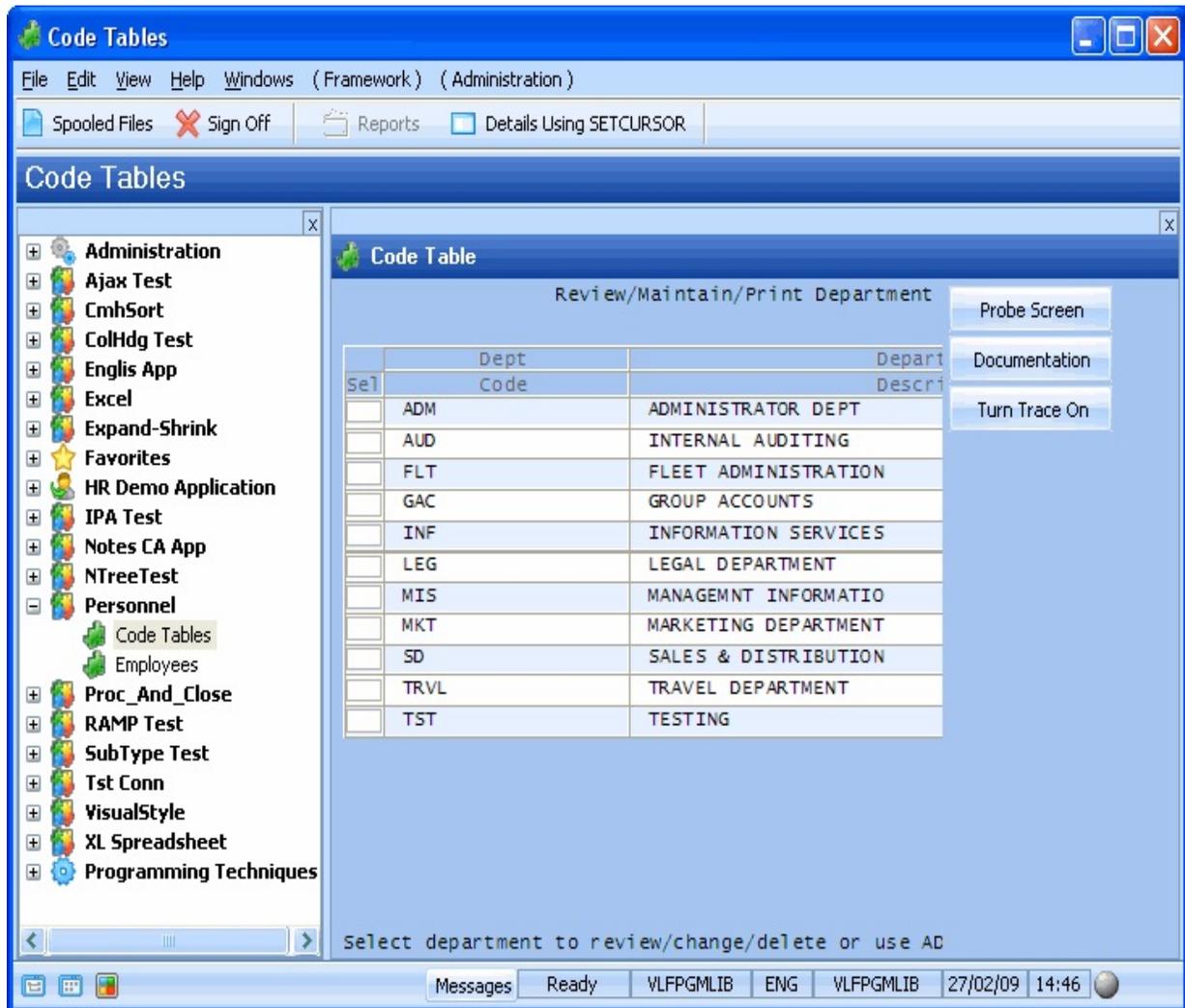


When working in RAMP Tools, you typically do a Partial Save of your work. Only when you are ready to test your application in end-user mode, you need to do a Full Save of the RAMP definitions.

4. Display the Framework and select the Personnel application and then Code Tables business object. The Personnel Table Main menu is displayed.



5. Select one of the options in the menu to access the code tables.



Note that your screen may look slightly different depending on how your system is set up.

The code tables maintenance application segment is now modernized and fully functional in the Framework. Note that depending on your RAMP-TS configuration your modernized screen may look slightly different.

Summary

Important Observations

You do not have to use RAMP on every single 5250 screen in your application. Often whole application segments are rarely used, and typically these can be snapped into a Framework in a very short time

You need to carefully consider how much time and resources you want to spend modernizing parts of an application.

You can do the naming at any time in any order and you don't have to name all the screens and fields used in navigation in a single session. However, you need to name all screens and fields before you classify them and track the application navigation.

What You Should Know

How to create an application and a business object in the Framework and how to associate command handlers with the business object.

How to name screens

How to classify screens and track navigation

How to enable and disable buttons on a screen

How to snap your screen into the Framework.

RAMP-TS003: Creating a Data Filter for Employees

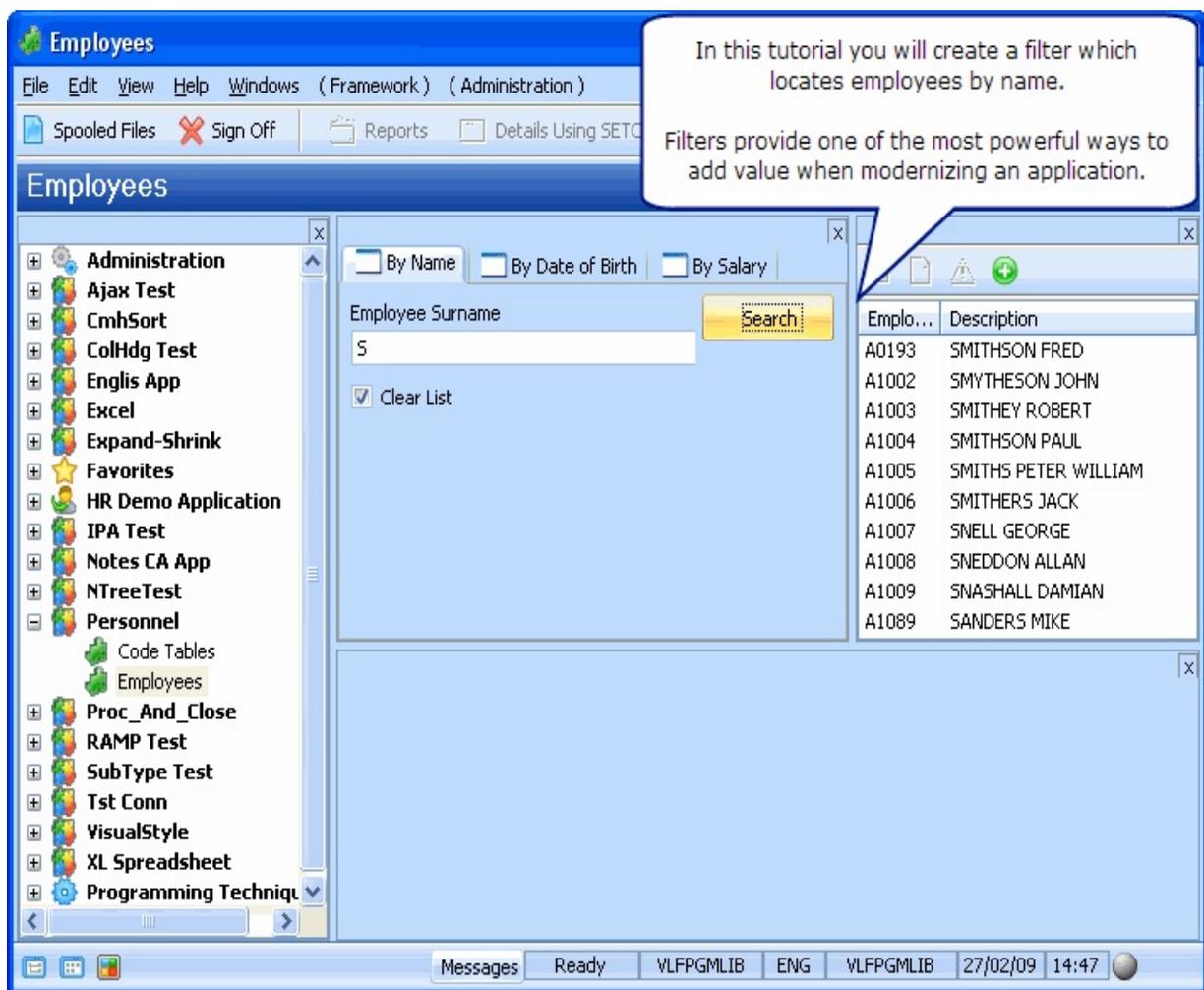
In this tutorial you will create a functional By Name filter for the Employees business object. You will use the Program Coding Assistant to create the filter.

Objectives

Understand how fundamentally filters add value to a modernized application

Learn how to quickly create filters to provide different ways of accessing information in the application

•



The screenshot shows the 'Employees' application window. The title bar reads 'Employees' and the menu bar includes 'File', 'Edit', 'View', 'Help', 'Windows', '(Framework)', and '(Administration)'. Below the menu bar, there are buttons for 'Spooled Files', 'Sign Off', 'Reports', and 'Details Using SETC'. The main window is titled 'Employees' and features a tree view on the left with various application components like 'Administration', 'Ajax Test', 'CmhSort', etc. The main area has three filter options: 'By Name', 'By Date of Birth', and 'By Salary'. The 'By Name' filter is active, showing a search box for 'Employee Surname' with the letter 'S' entered and a 'Search' button. A 'Clear List' checkbox is also present. A table on the right displays the results of the search:

Empl...	Description
A0193	SMITHSON FRED
A1002	SMYTHESON JOHN
A1003	SMITHEY ROBERT
A1004	SMITHSON PAUL
A1005	SMITHS PETER WILLIAM
A1006	SMITHERS JACK
A1007	SNELL GEORGE
A1008	SNEDDON ALLAN
A1009	SNASHALL DAMIAN
A1089	SANDERS MIKE

Two callout boxes provide additional context: one states 'In this tutorial you will create a filter which locates employees by name.' and another states 'Filters provide one of the most powerful ways to add value when modernizing an application.'

To achieve this objective, you will complete the following steps:

[RAMP-TS003 Step 1. Creating Your Real By Name Filter](#)

[RAMP-TS003 Step 2. Snapping In the By Name Filter](#)

[RAMP-TS003 Step 3. Filter Code](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

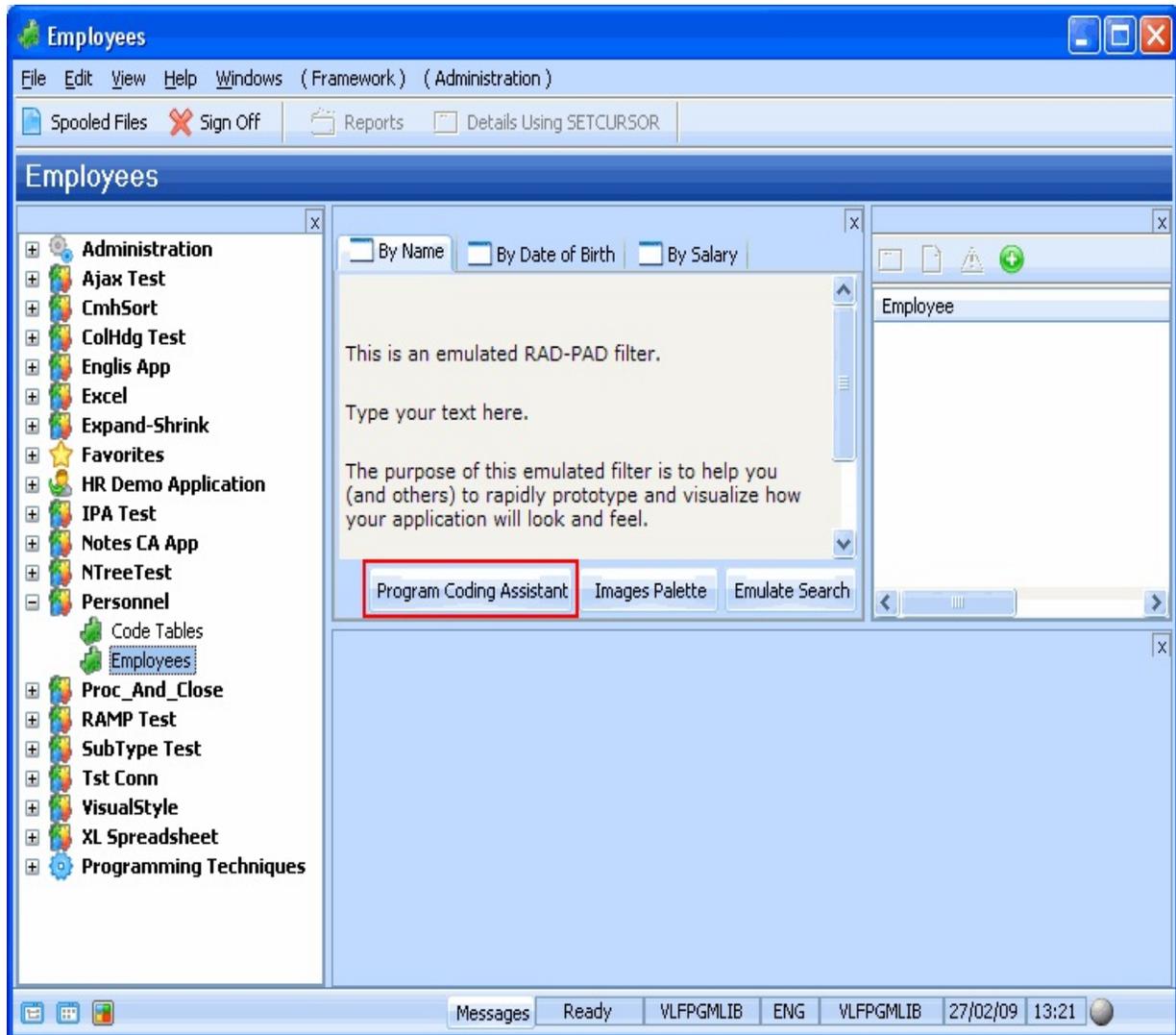
[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

RAMP-TS003 Step 1. Creating Your Real By Name Filter

In this step, you will create a real filter which searches the PSLMST file by employee surname. You will also learn how to use the Program Coding Assistant.

1. Click on the Program Coding Assistant button in the By Name filter.



The Program Coding Assistant window is displayed. It allows you to create different types of components that can be plugged into your filters, instance lists and command handlers. It is highly recommended to use the program coding assistant when you first start using the Framework.

Initially you will most likely use filters that generate a component that can be executed (e.g. CRUD Filter (Create/Read/Update/Delete), Filter that searches a file or view). As you progress you might only use a skeleton filter or simply copy from one that is similar to one that you want to create.

2. If you are using a non-English system, click on Framework -> Your Framework in the top-left tree view. The Set LANSA code generation preferences option appears at the bottom. Select this option and set your preferences.
3. In the list on the top left, ensure the Personnel application, Employees and the By Name filter are selected.
4. Underneath it, select Native MS Windows as the platform.
5. As the type of code you want to generate, select Filter that searches using a file or view.

Program Coding Assistant

Select the object you want to generate code for

- Application->Tst Conn
- Application->Personnel
 - Business Object->Code Tables
 - Business Object->Employees
 - Command Handler->Details
 - Command Handler->New
 - Command Handler->Documents
 - Command Handler->Events
 - Filter->By Name**
 - Filter->By Date of Birth
 - Filter->By Salary

Refresh

Select the platform you want to generate for

- Native MS Windows**
- Web - using *WEBEVENT functions
- Web - using WAM components
- Web - using AJAX style components

Select the type of code you want to generate

Filter that searches by all logical views of a file

CRUD Filter

Filter that searches using a file or view

A skeleton filter

Search button event handling routine (code fragment)

Invoke #avListManager.AddtoList (code fragment)

Filter that searches using a file or view

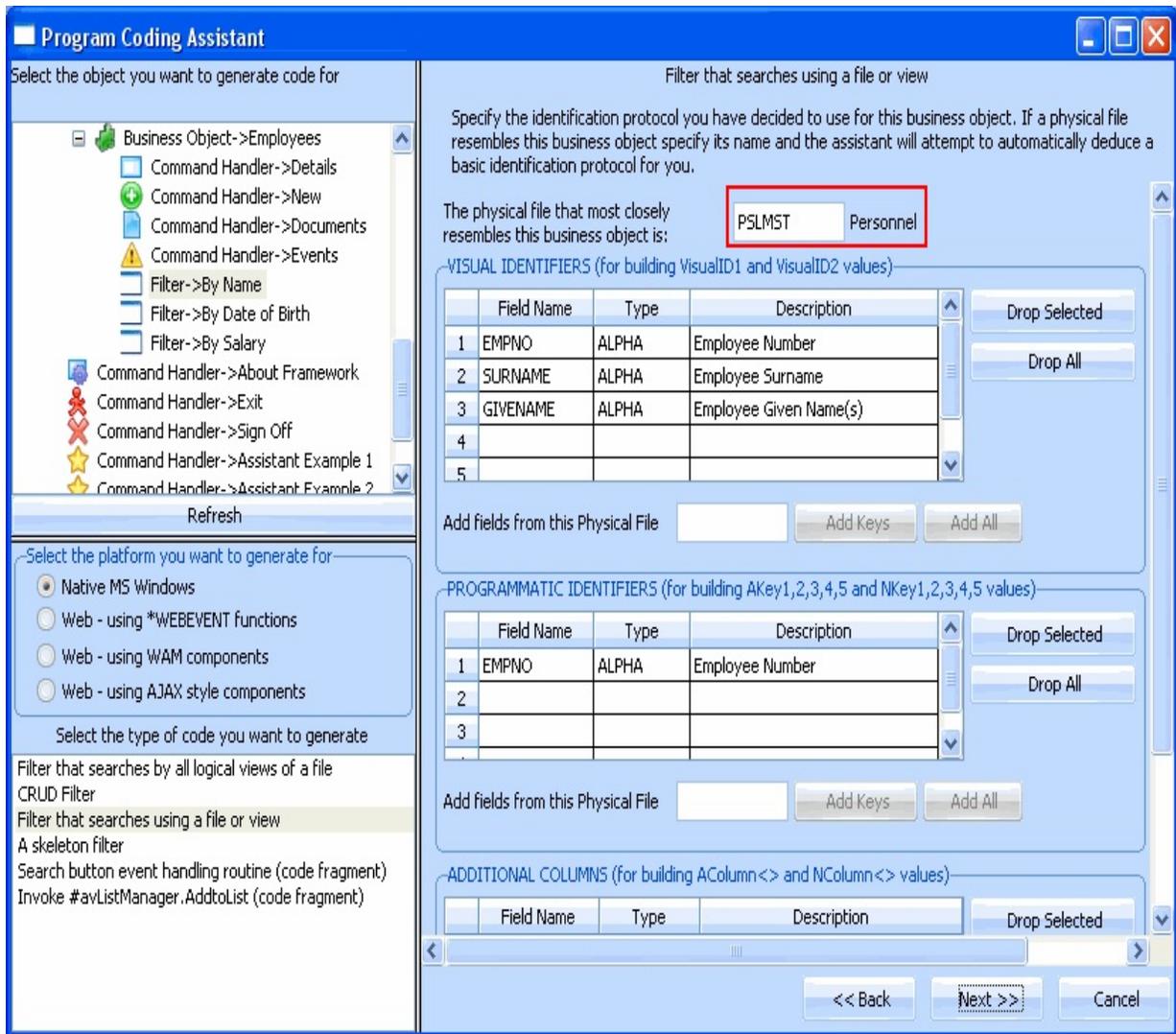
What? This assistant produces the code for a filter that searches information using a specified physical file or logical view.

Filters are used to dynamically create business object *inst lists* (e.g. lists of Customers, lists of Products, lists of Ord lists of Employees, etc).

Typically Visual LANSA Framework filters are presented like example (in the area circled in red):

Next >> Cancel

- Click the Next button.
- On the next page specify PSLMST as The physical file that most closely resembles this business object.



The Program Coding Assistant detects the Visual and Programmatic Identifiers required:

A Visual Identifier is the field or fields that a user would use to identify a unique instance of the business object.

A Programmatic Identifier is the field(s) that the program would use to identify a unique instance of the business object. Typically these would be the primary keys of the file or files that make up the data in the instance list.

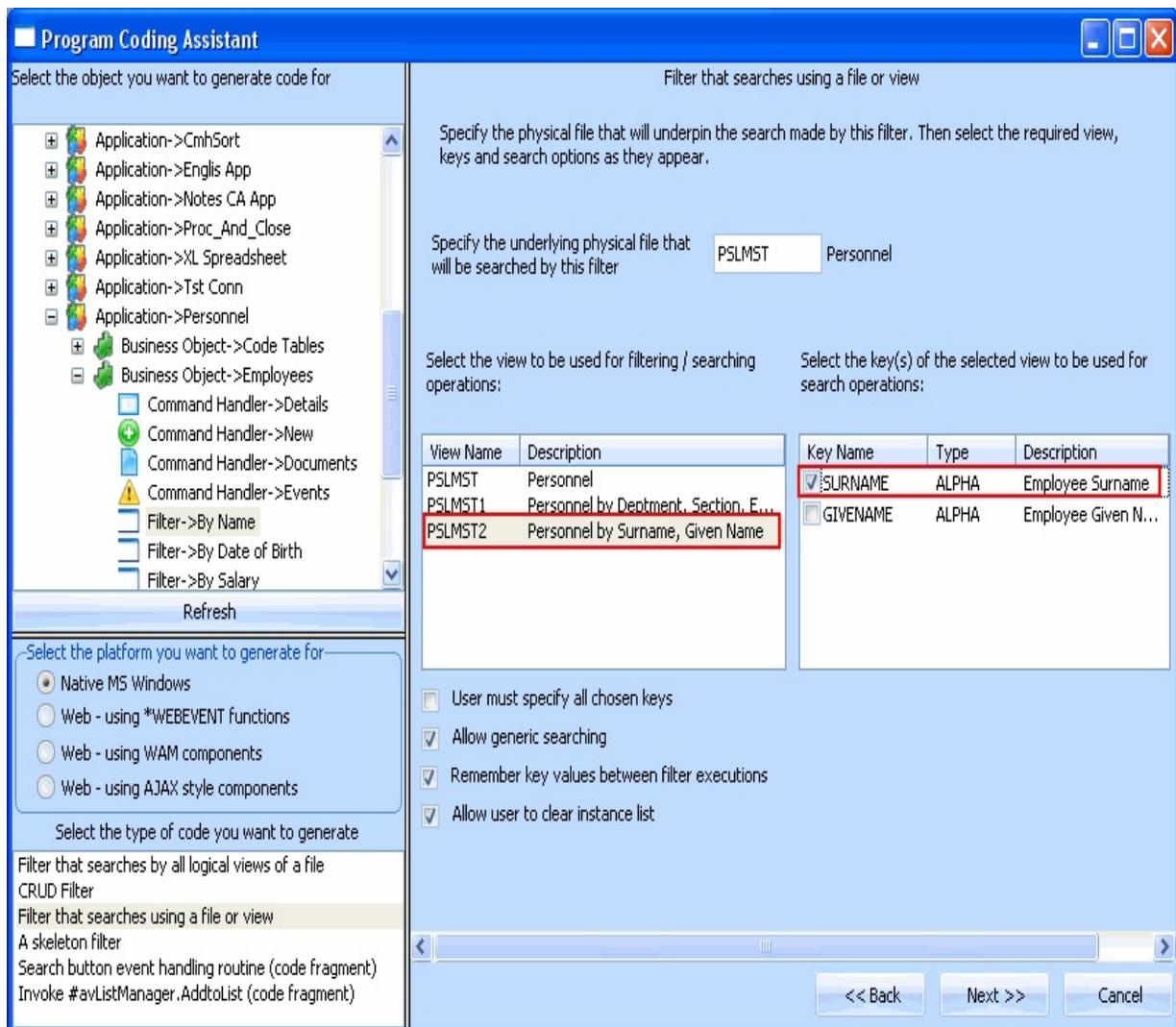
The additional columns represent the additional columns in your instance list that you may have added during the prototyping phase.

8. Click the Next button.

9. On the next page specify PSLMST2 as the view to be used for filtering/searching operations. It is logical view of the PSLMST file keyed by the SURNAME and GIVENAME fields.

Note that you need an appropriate logical file for each filter that you want to create. Before implementing all your filters, review your data model to confirm that all the logical files exist. Doing so will speed up the process of implementing your prototype.

10. Select the SURNAME field as the key of the view to be used for search operations.

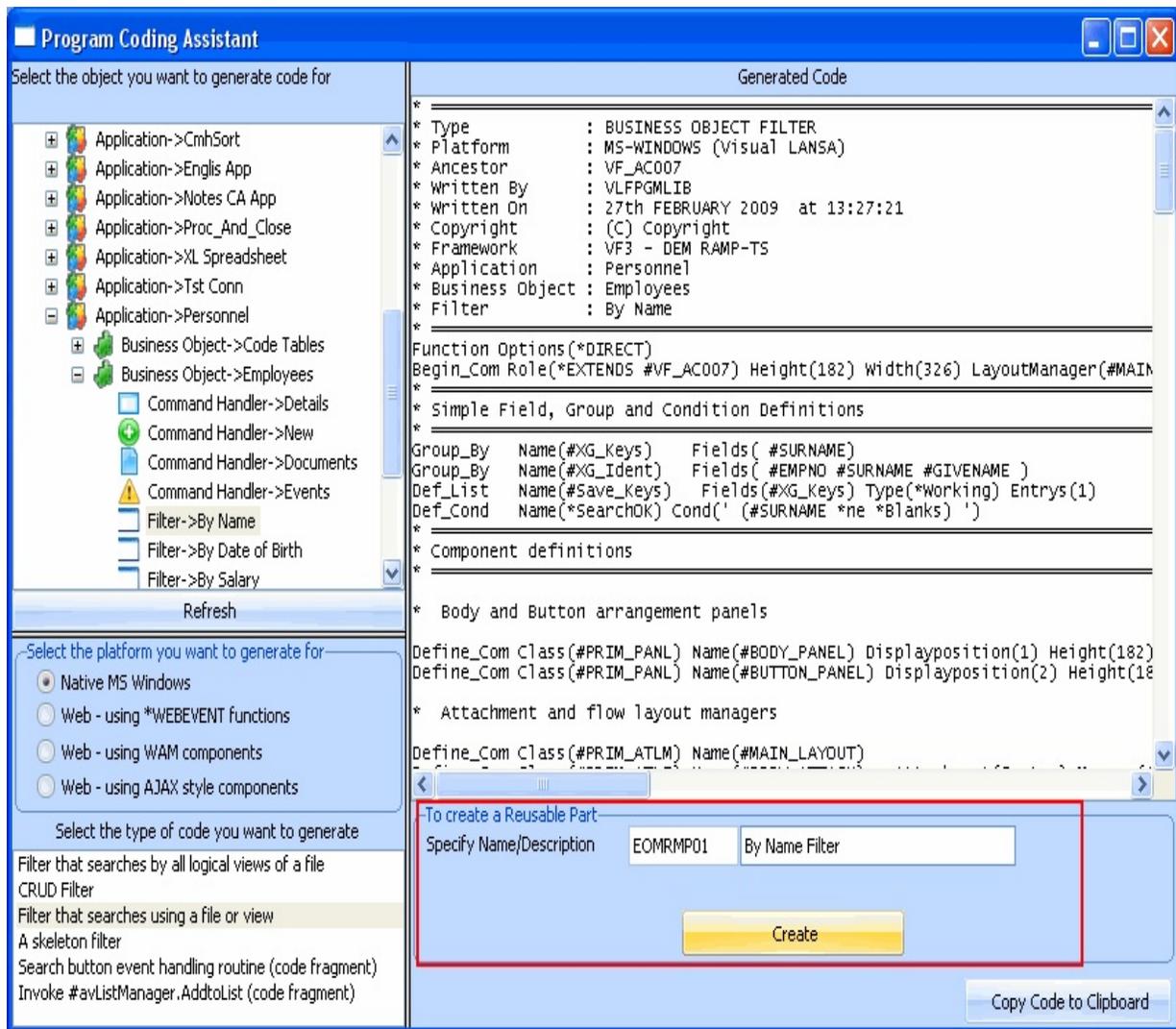


11. Click the Next button.

12. Select the option Routine to listen for signals to update the instance list.
13. Click the Generate Code button.

The next page, Generated Code, displays the source code for your filter. You now need to create the component that will contain this code:

14. Specify iiiRMP01 as the name of your real filter and By Name Filter as the description. (iii are your initials If you are using an unlicensed or trial version of Visual LANSA, you must always use the 3 characters DEM to replace iii).
15. Click on the Create button to create the component.



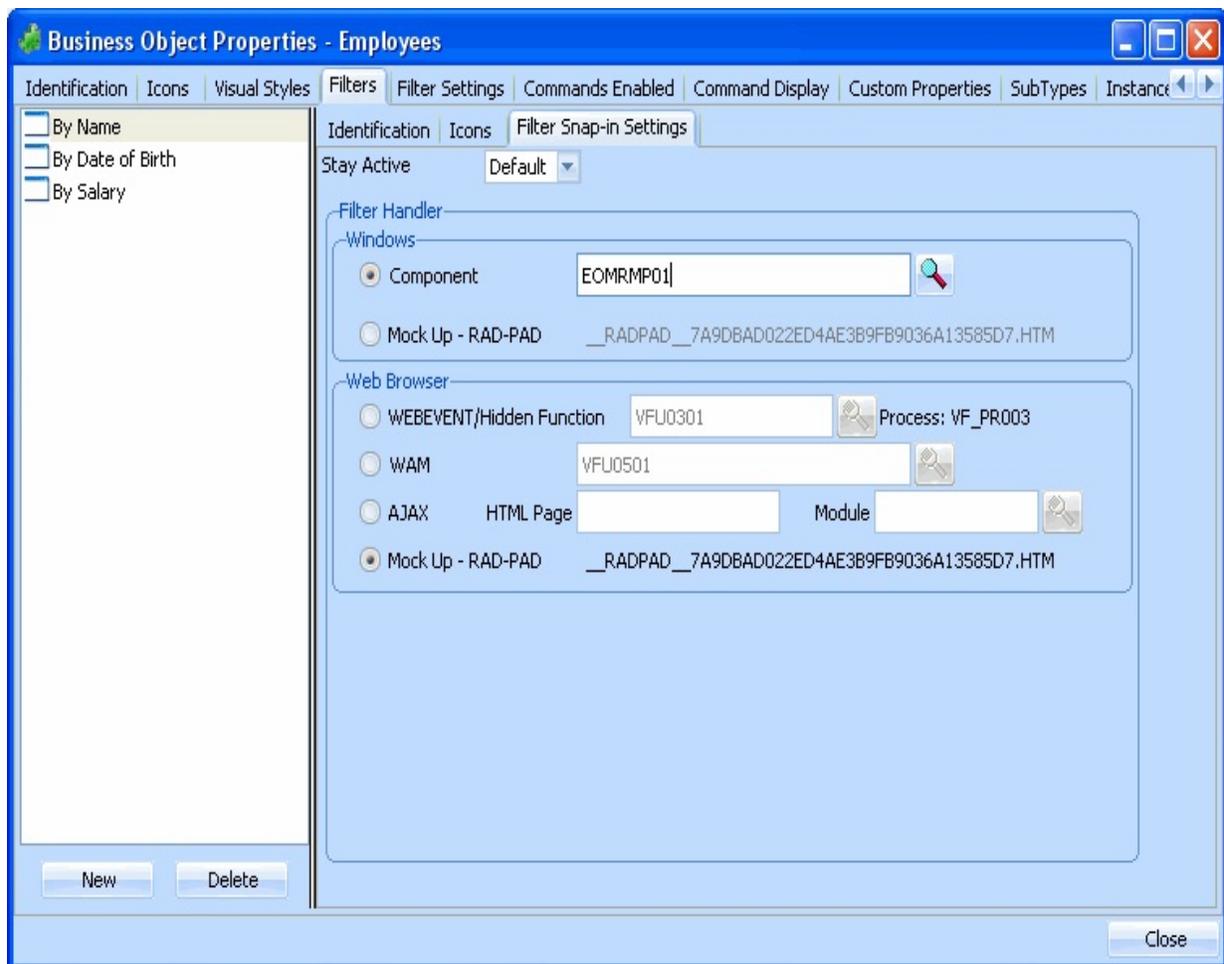
After a brief delay the Filter component is displayed in the Visual LANSA editor.

16. Compile the component.

RAMP-TS003 Step 2. Snapping In the By Name Filter

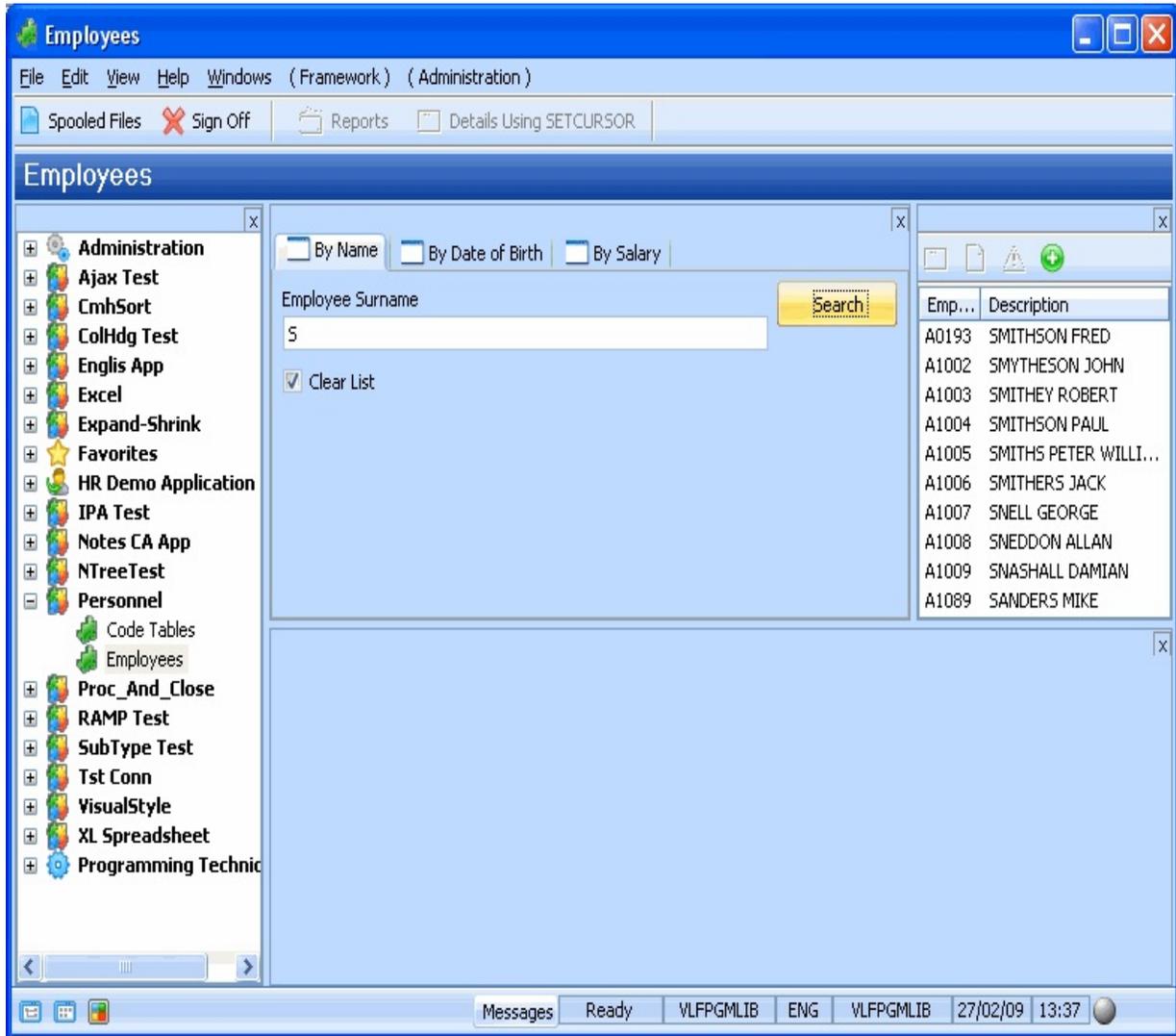
Now that you have compiled your filter and are ready to test it, you need to snap it into the Framework.

1. In the Framework, close the Program Coding Assistant.
2. Double-click on the Employees business object to display its properties.
3. Display the Filter Snap-in Settings tab.
4. Specify iiiRMP01 as the Windows filter handler component.



5. Close the Employees business object properties and display the By Name filter. You can now see your real filter.
6. Type in a letter in the Surname field and click on the Search button to verify

that your real filter has been snapped in the Framework and is usable.



RAMP-TS003 Step 3. Filter Code

Even though you can create most filters simply by using the Program Coding Assistant, you should understand how they are coded.

1. Switch to the Visual LANSAs editor where the iiiRMP01 reusable part is open.
2. Review the generated source code in the Source tab to see how the filter is coded to add data to the instance list:

The Framework is notified that an update is about to occur.

Invoke `#avListManager.BeginListUpdate`

Next, the list is cleared of any existing items.

Invoke `#avListManager.ClearList`

Next, data is selected. You can use one of the techniques you learnt in the Visual LANSAs Fundamentals tutorials to do this. For example:

```
Select Fields(#XG_Ident) From_File(PSLMST2) With_key(#XG_Keys)
Generic(*yes) Nbr_Keys(*Compute)
```

Next, the visual identifiers are set up:

```
Change #UF_VisID1 #EMPNO
```

```
Change #UF_VisID2 #SURNAME
```

Then the data is added to the list.

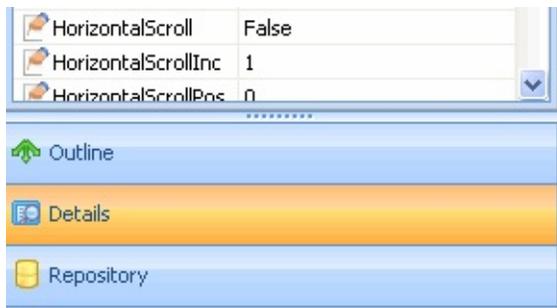
```
Invoke #avListManager.AddtoList Visualid1(#UF_VisID1)
Visualid2(#UF_VisID2) AKey1(#EMPNO)
```

VisualId1 will be shown in column one of the instance list and VisualId2 will be shown in column two of the instance list. Akey1 is the key that uniquely identifies an employee (in this case the field is alphanumeric, so its Akey1, not Nkey1).

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

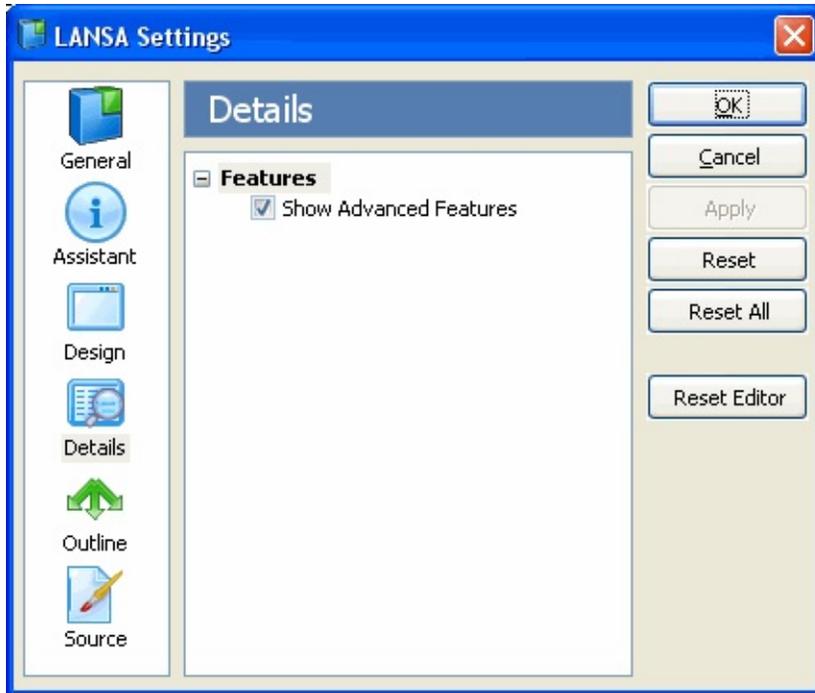
```
Invoke #avListManager.EndListUpdate)
```

3. Next click on Details tab in the editor to display the properties of your component.

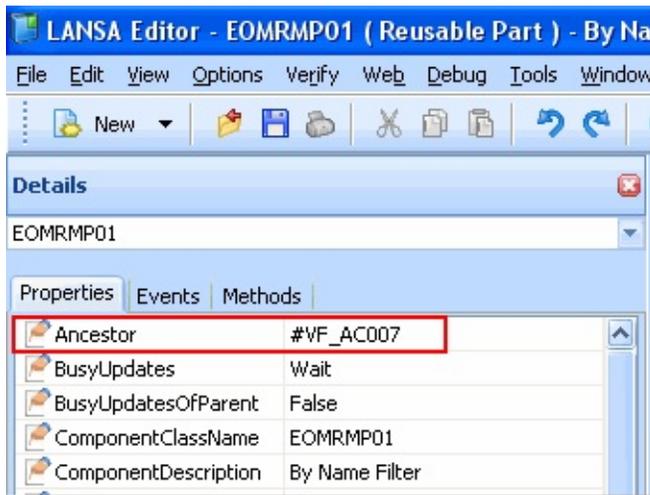


You need to ensure that all properties are displayed:

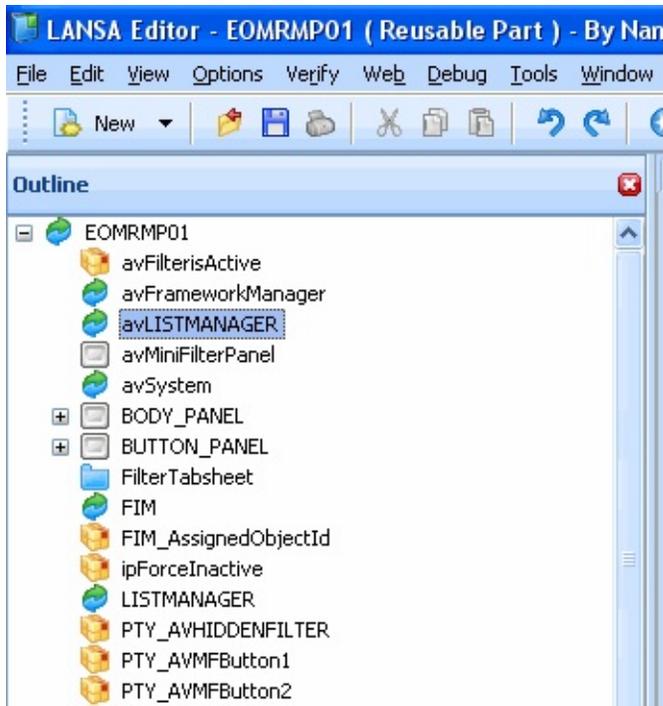
4. Select the Settings option in the Options menu.
5. Click on Details and make sure the Show Advanced Features option is selected.



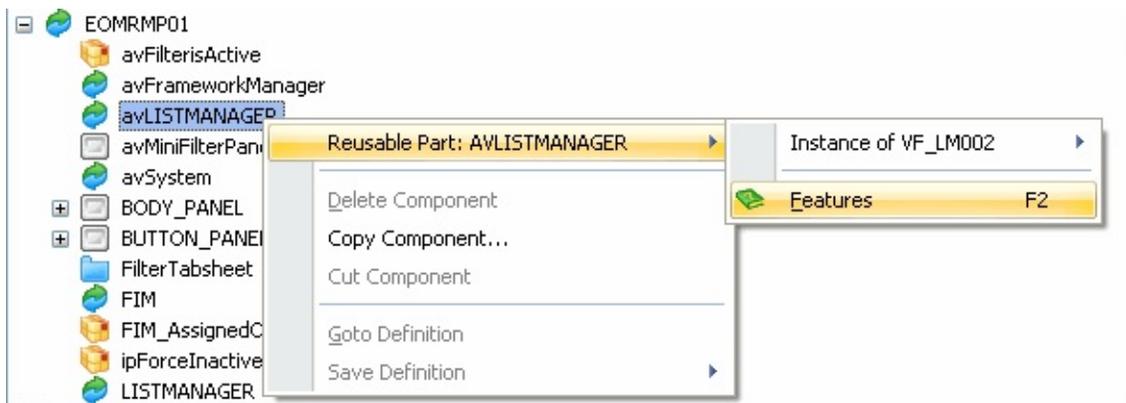
6. Notice that the Ancestor property of the component is #VF_AC007. All filters inherit from this base class which provides a set of predefined behavior.



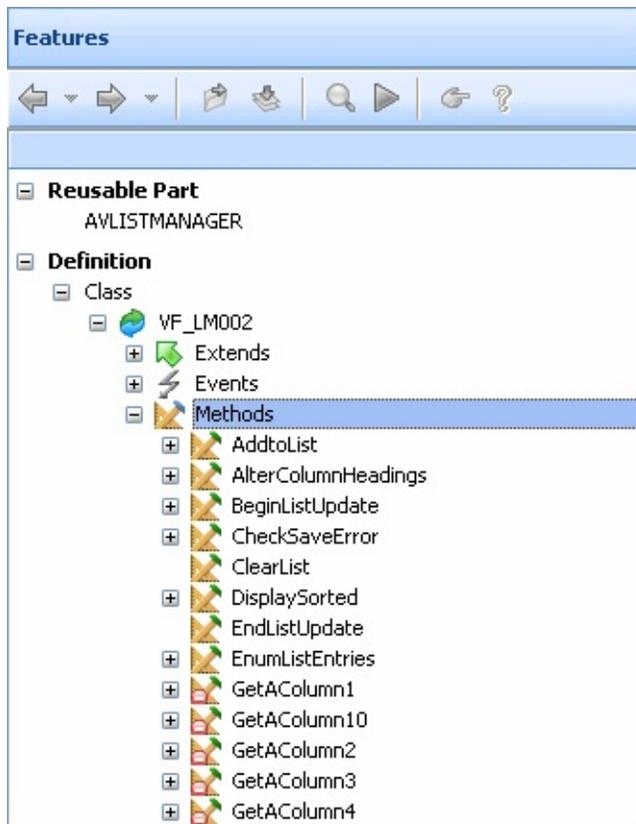
7. Click on the Outline tab in the editor to see what components you inherit from the VF_AC007 ancestor component.



8. Right-click the avLISTMANAGER component and select the Features option.



9. Expand the methods of the component and examine them.



10. Close the iiiRMP01 component.

You may want to read Filter and Command Handler Anatomy in the Framework guide to see how these components are structured.

Summary

Important Observations

Filters are one of the main ways of adding value in a modernized application.

-

Tips & Techniques

The source code for the filters used in the demonstration application can be found in the repository in components named DF_*

What I Should Know

What you need to do to create your own filters.

How you snap them in the Framework.

How to use the Program Coding Assistant.

RAMP-TS004: Naming and Classifying the Employee Screens

In this tutorial you will name and classify the Personnel System screens that will be used in the Employees business object.

Objectives

Learn more about naming and classifying screens.



In this tutorial you will track the basic navigation in the Personnel System application segments you are going to modernize.

You will also classify the screens and take snapshots of the destination screens which will be used to refine your prototype in the next tutorial.

To achieve this objective, you will complete the following steps:

[RAMP-TS004 Step 1. Name the Screens](#)

[RAMP-TS004 Step 2. Classify Screens](#)

[RAMP-TS004 Step 3. Track Navigation](#)

[RAMP-TS004 Step 4. Take Snapshots of Your Destination Screens](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

RAMP-TS001: Creating a Basic Prototype of the Modernized Application

RAMP-TS002: Rapidly Modernizing Complete Application Segments

RAMP-TS003: Creating a Data Filter for Employees

RAMP-TS004 Step 1. Name the Screens

In this step you will name the screens and the required fields in your application.

1. Start the Framework.
2. Start RAMP.
3. Connect to the server.
4. Start the PLSYS application by entering the command:

```
lansa run pslsys partition(dem)
```

5. Name the main menu PLSYSMain and the option field txtOption:

The screenshot shows the 'Screens' tab in the RAMP application configuration tool. The 'Screen Definition' section is active, showing the following details:

- Name:** PLSYSMain (ID = 5)
- Variant Name:** (empty)
- Description:** (empty)
- Subfiles:**

Start	End	Name	Lines/Entry
5	19	SFL004SF	1
- Fields:**

Name	Row	Col	Sfl
Personnel System Main Menu	1	20	
	3	2	
Enter number of function required or place	5	2	Y
	5	5	Y
1.	5	9	Y
Full Employee Listing	22	2	

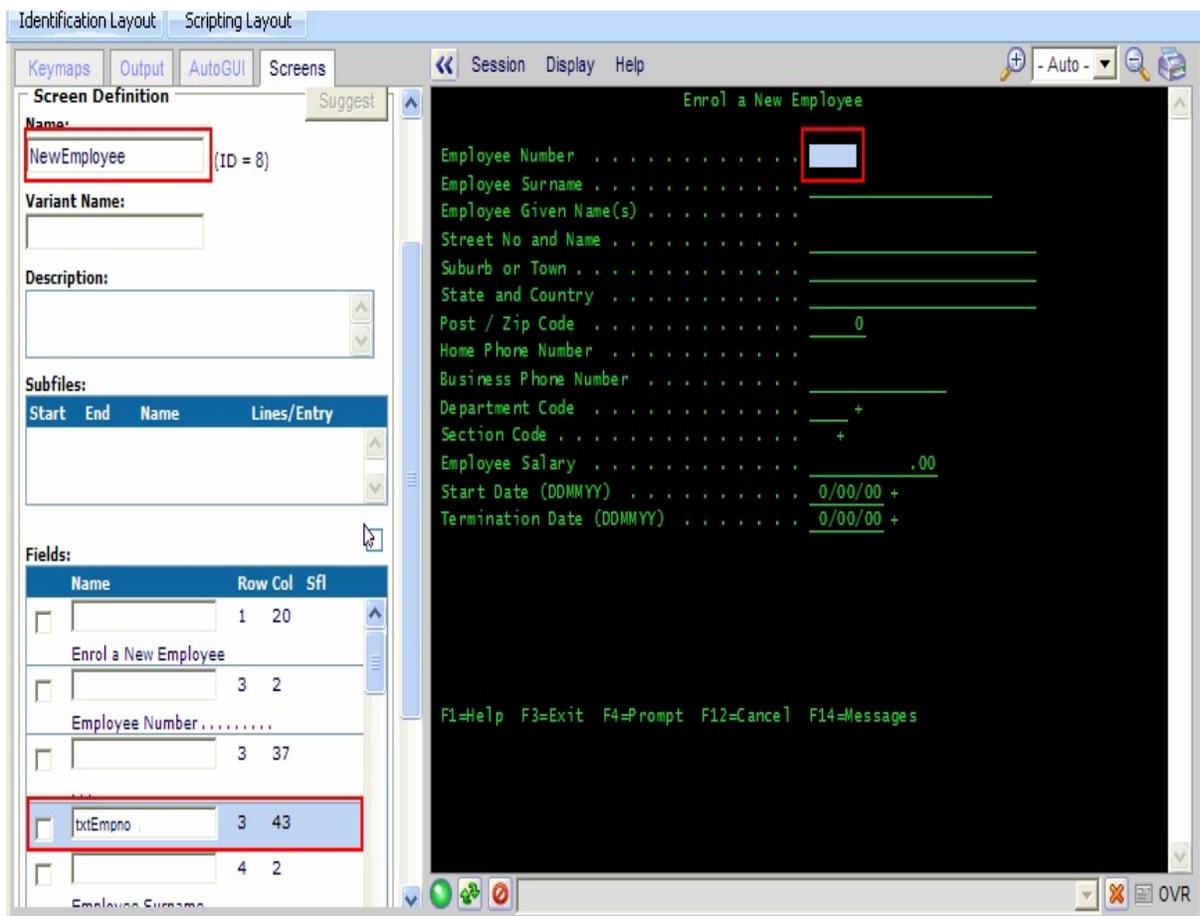
The 'Name' field in the 'Screen Definition' section and the 'txtOption' field in the 'Fields' table are highlighted with red boxes.

Note that you should not select the check the box in front of a field. The checkbox is used to use the field to differentiate between otherwise identical screens.

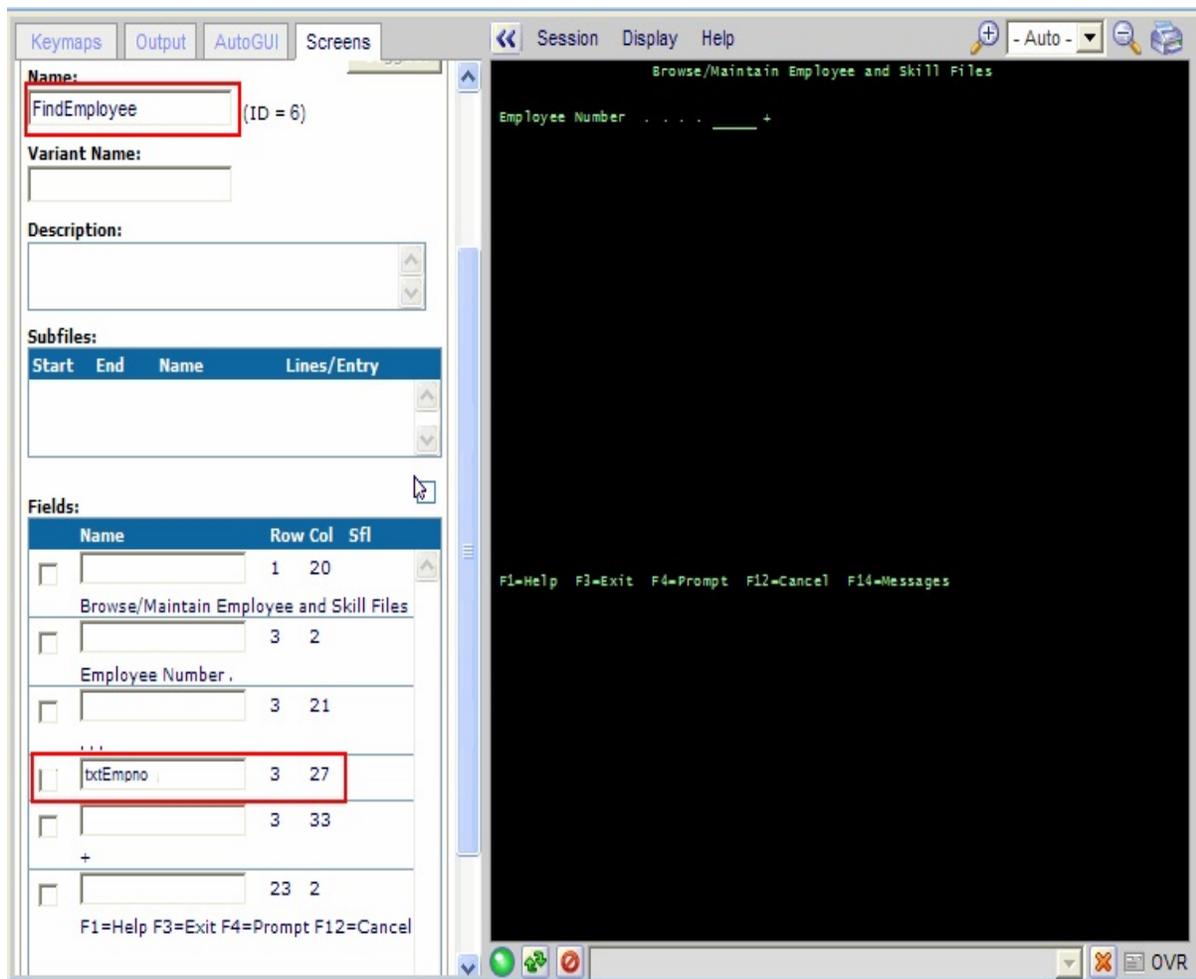
In the following steps, whenever a message appears asking you if you want to save the changes you have made, click OK.



6. Select option 2 in the menu to display the Enrol a New Employee screen.
7. Name it NewEmployee
8. Name the Employee number field txtEmpno and click Save.

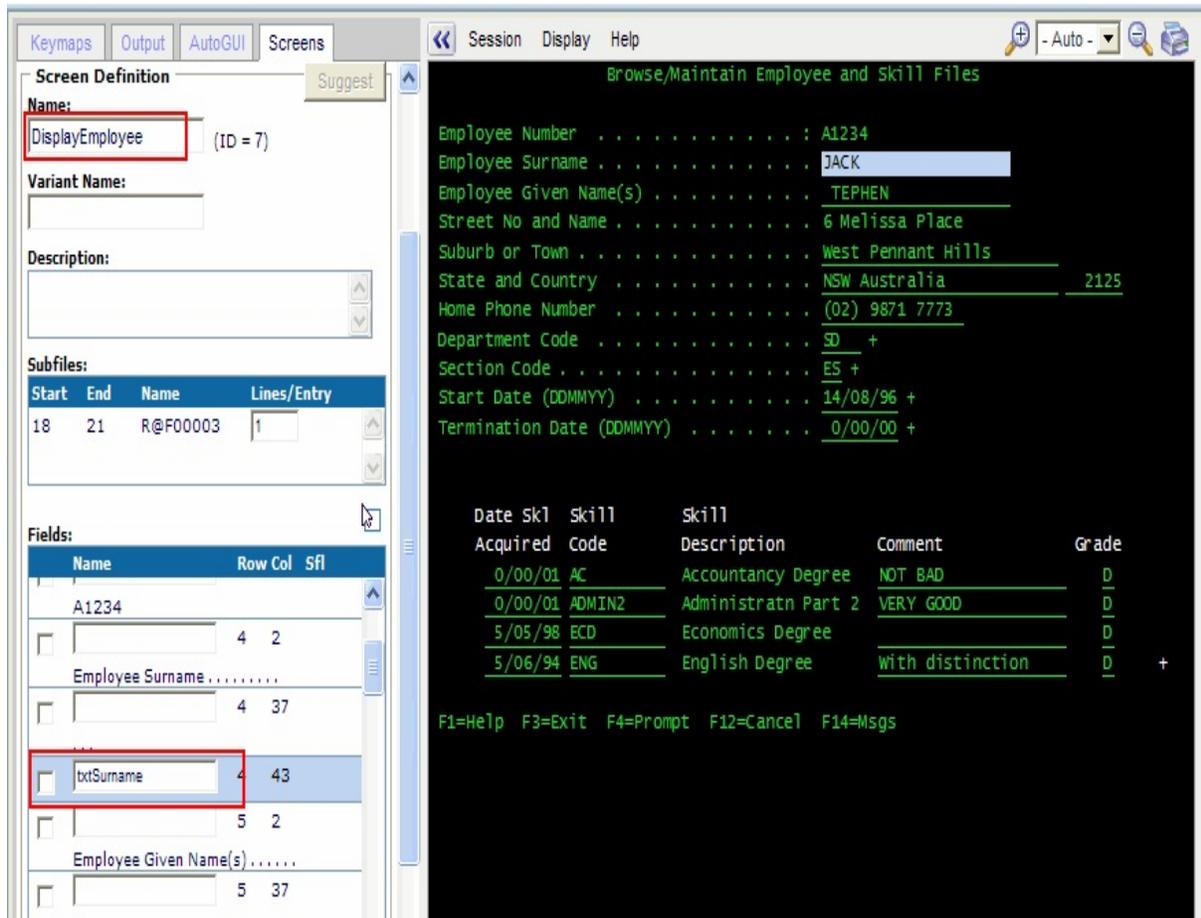


9. Press F12 to return to the Personnel System Main Menu.
10. Select option 3 in the menu to display employee and skills details.
11. Name the employee search screen FindEmployee.
12. Locate the Employee Number field and name it txtEmpno:

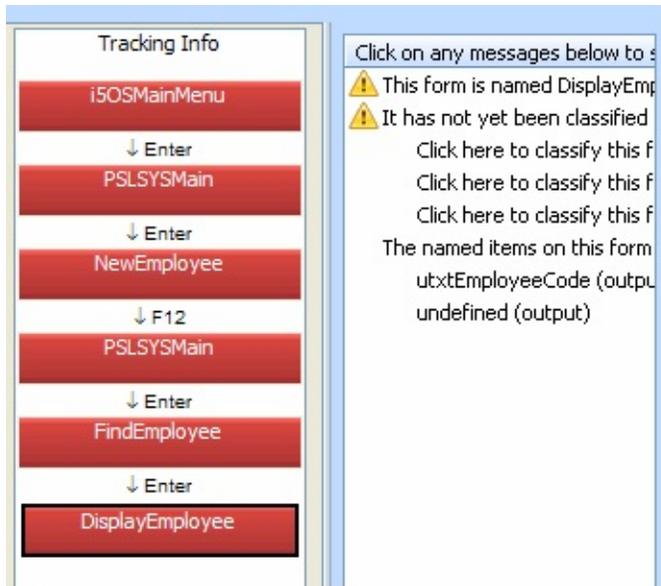


13. Click Save.
14. Type in any employee number, for example A1234, and press Enter to display the Browse/Maintain Employee and Skill Files screen.
15. Name the screen DisplayEmployee.
16. Click on the Save button.
17. Press F21 to display the screen in update mode so that you can name fields.

18. Name the Surname field on the screen txtSurname.



You have now finished naming your screens:

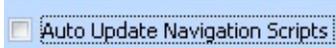


19. Do a partial save of your RAMP definition.
20. Return to the i5/OS Main Menu by pressing F3.

RAMP-TS004 Step 2. Classify Screens

In this step you will classify the screens as [Junction Screen](#) , [Special Screen](#) or [Destination screen](#).

1. Deselect the Auto Update Navigation Scripts check box (you will track navigation in the next step, now you are just classifying the screens) and ignore any messages about auto navigation for the moment:



2. Log on to your system in the RAMP-TS 5250 session.
3. If the messages screen is displayed dismiss it.
4. In the i5/OS Main menu, enter this command:

```
lansa run pslsys partition(dem)
```

5. Select option 2 to enrol a new employee.
6. When the Enrol a New Employee screen is displayed, press F12 to return to the Personnel System main menu.
7. Select option 3 to browse employees.
8. In the Find Employee screen type any employee number, for example A1234.

The Browse/Maintain Employee and Skill Files screen is displayed.

You now have a list of all the screens in the Personnel System application about to be modernized ready to be classified (you can do the classification either while you are tracking the navigation as in [RAMP-TS002 Step 2: Classify the Screens and Track Navigation](#) or you can do it as two distinct steps as in here).

It is usually preferable to make the classification of screens a distinct task.

When you are ready, you can use your spreadsheet or whatever document you have and methodically track the movement through the junctions to your destinations. Doing it this way should produce the least amount of rework later.

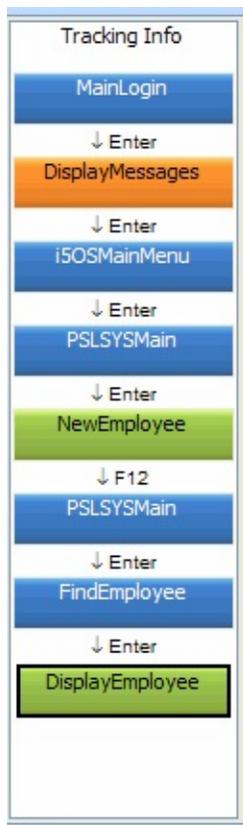
Now you can start classifying the screens:

9. Click on the MainLogin screen in the Tracking Info area. Notice that it is predefined as a Junction (it is automatically classified).

10. Using the Tracking Info area, classify the rest of the screens like this:

- PSLSYSMain Junction
- NewEmployee Destination
- FindEmployee Junction
- DisplayEmployee Destination

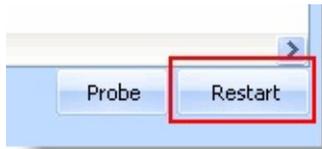
You have now completed classifying the screens.



RAMP-TS004 Step 3. Track Navigation

In this step you will track the navigation in the application.

1. Cancel back to the i5/OS Main Menu and sign off.
2. Clear the tracking information by clicking the Restart button:



3. Select the Auto Update Navigation Scripts check box.



4. Log on to your system in the RAMP-TS 5250 session.
5. If the messages screen is displayed dismiss it by pressing Enter.
6. In the i5/OS Main menu, enter this command:

```
lansa run pslsys partition(dem)
```

7. Select option 2 to enrol a new employee.
8. When the Enrol a New Employee screen is displayed, press F12 to return to the Personnel System main menu.
9. Select option 3 to browse employees.
10. In the Find Employee screen type any employee number, for example A1234.
11. Return from the Browse/Maintain Employee and Skill Files screen to the Personnel Main menu by pressing F12.
12. Press F12 to return to the i5/OS Main menu.
13. Sign off.

You have now tracked the basic navigation in the Personnel System application

segments about to be modernized.

14. Deselect the Auto Update Navigation Scripts check box.

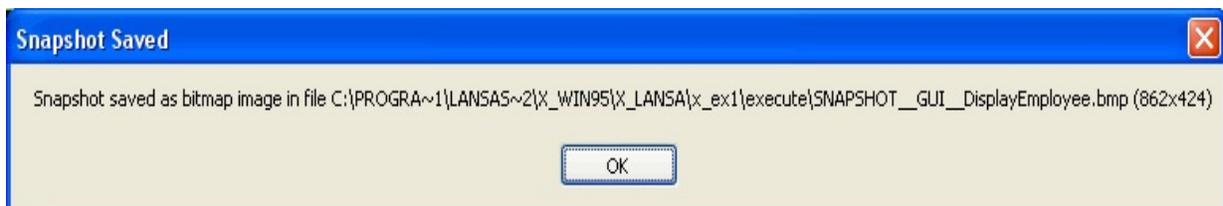
RAMP-TS004 Step 4. Take Snapshots of Your Destination Screens

In this step you will take snapshots of your two destination screens NewEmployee and DisplayEmployee. These snapshots will be used in the next tutorial to enhance your prototype.

1. Logon and navigate to the NewEmployee screen.
2. Make sure that the tab sheet (Screens, AutoGUI, Keymaps) on the left of the RAMP-TS 5250 emulator is hidden.
3. Click on the Snapshot button:



4. Note the message indicating that the snapshot is saved in your partition execute directory:



5. Take a snapshot of the DisplayEmployee screen.
6. Do a partial save of the RAMP information.

Summary

Important Observations

Before you track navigation, you need to name all the screens in your application and all the fields used for navigation.

You can classify screens either at the same time as you track navigation or when you have finished tracking.

You can take snapshots of your 5250 screens to be used in the prototype.

What You Should Know

How to name screens and navigation fields.

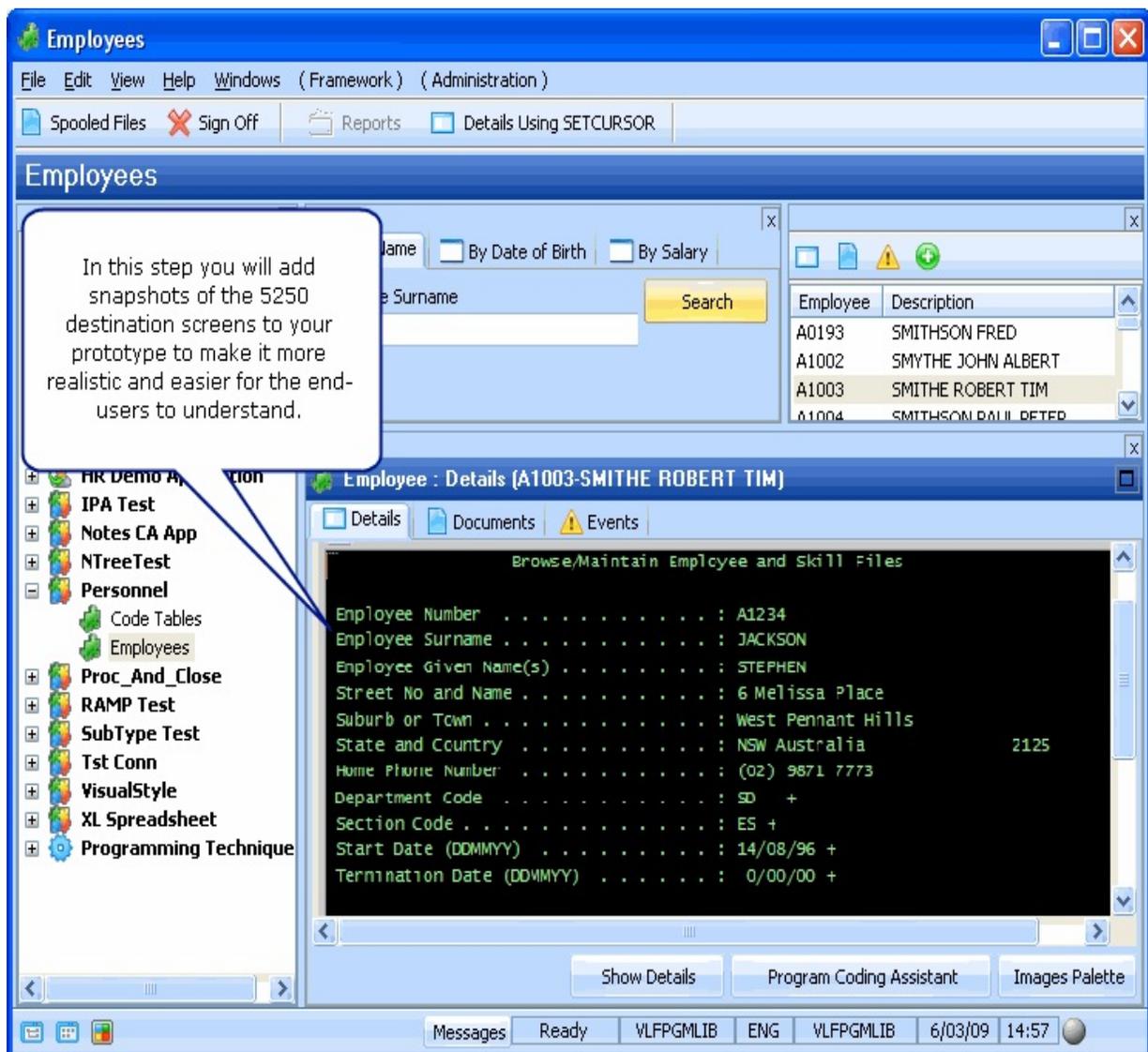
How to track navigation and classify screens.

RAMP-TS005: Reviewing Design

In this tutorial you will use snapshots of your destination screens to enhance your prototype.

Objectives

Learn how to add snapshots to the prototype



To achieve this objective, you will complete the following steps:

[RAMP-TS005 Step 1. Place Snapshots on Mock Up Command Handlers](#)

[RAMP-TS005 Step 2. Review the Prototype](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

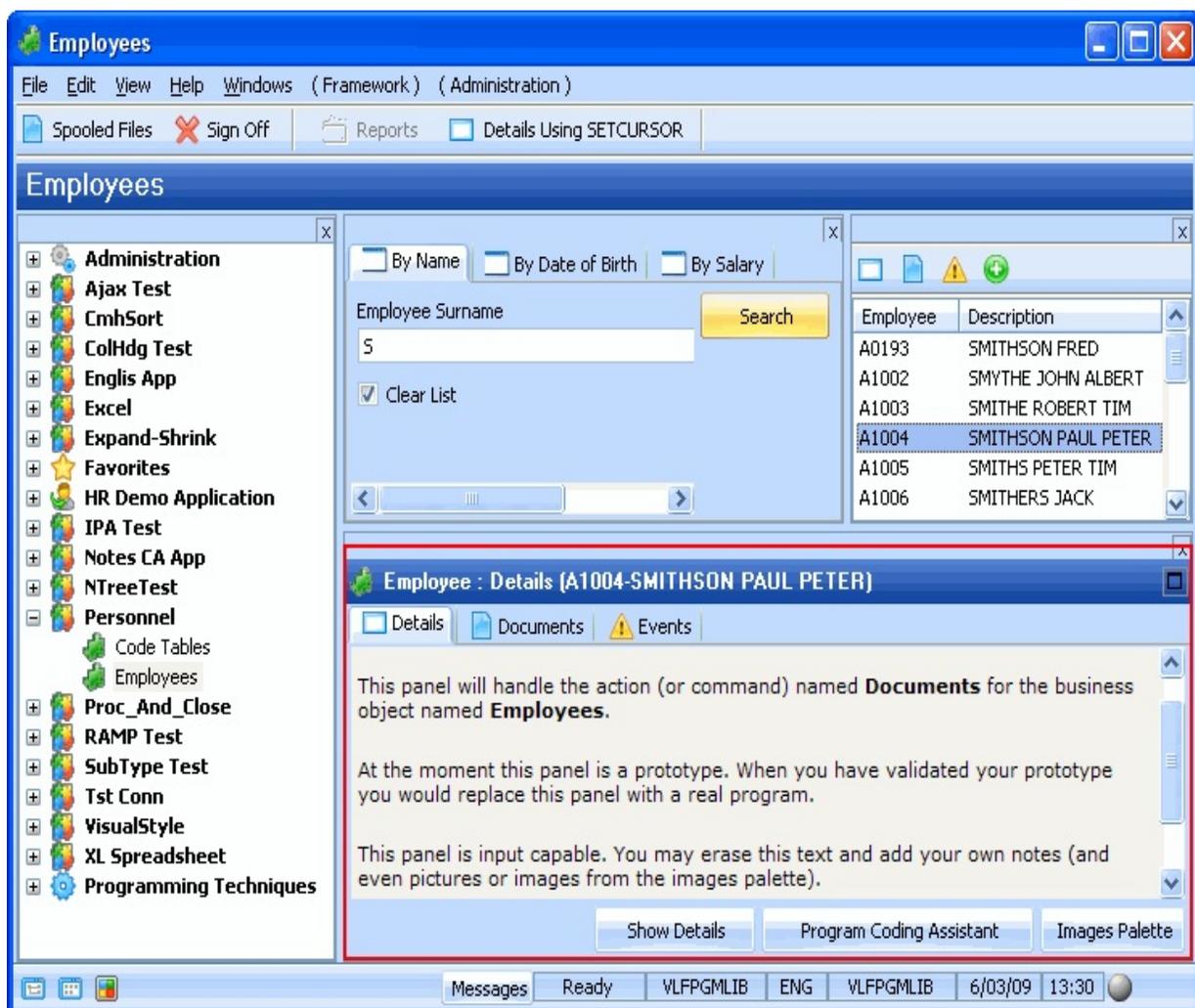
[RAMP-TS004: Naming and Classifying the Employee Screens](#)

RAMP-TS005 Step 1. Place Snapshots on Mock Up Command Handlers

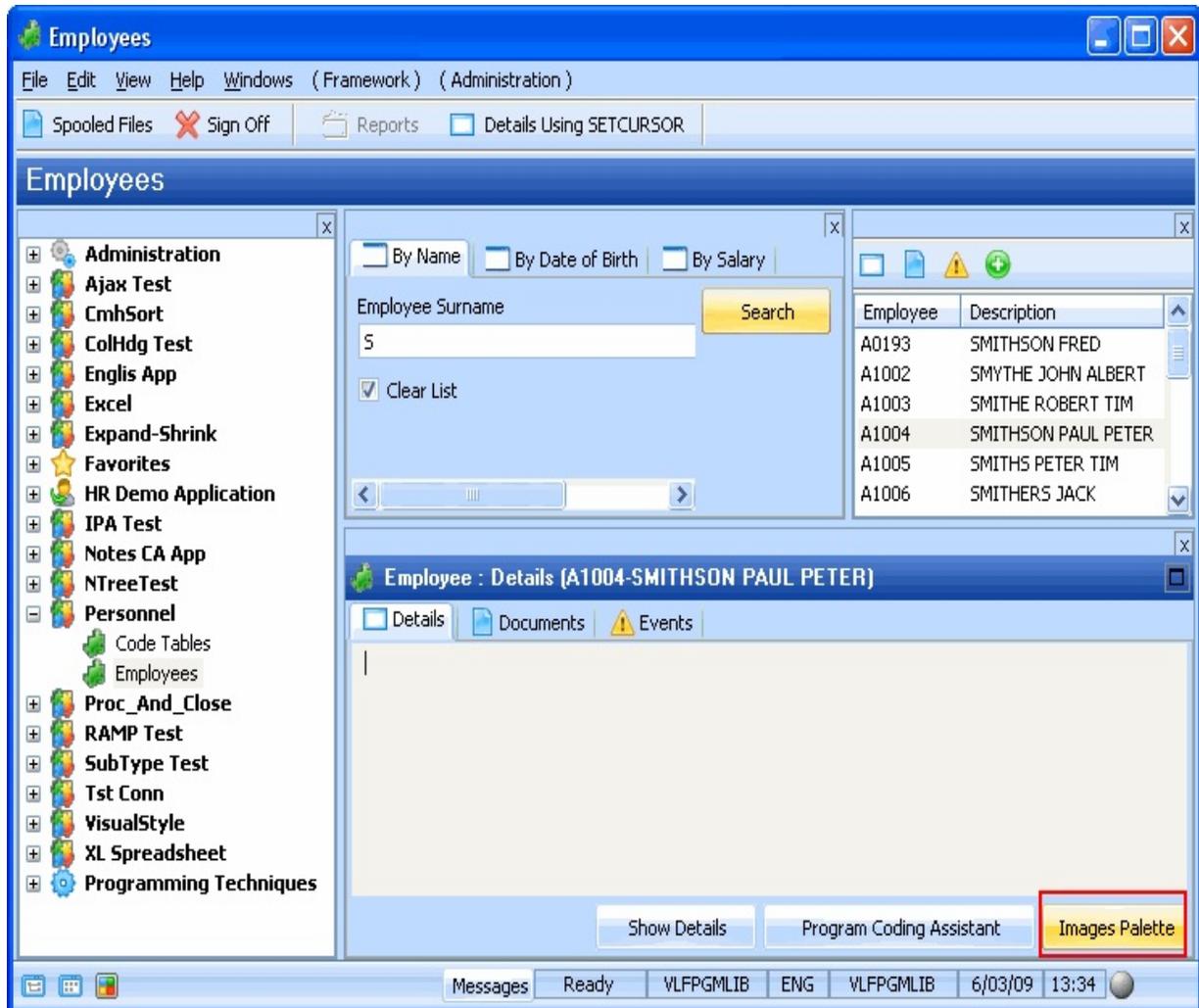
In the initial prototype, you created an executable model of the modernized application, with the main focus on how the end-users would access information. Now that you have named and classified the screens in the 5250 application, you can attach snapshots of your destination screens to the prototype to make it more visually complete.

In this step you will place snapshots of the 5250 destination screens on the command handlers.

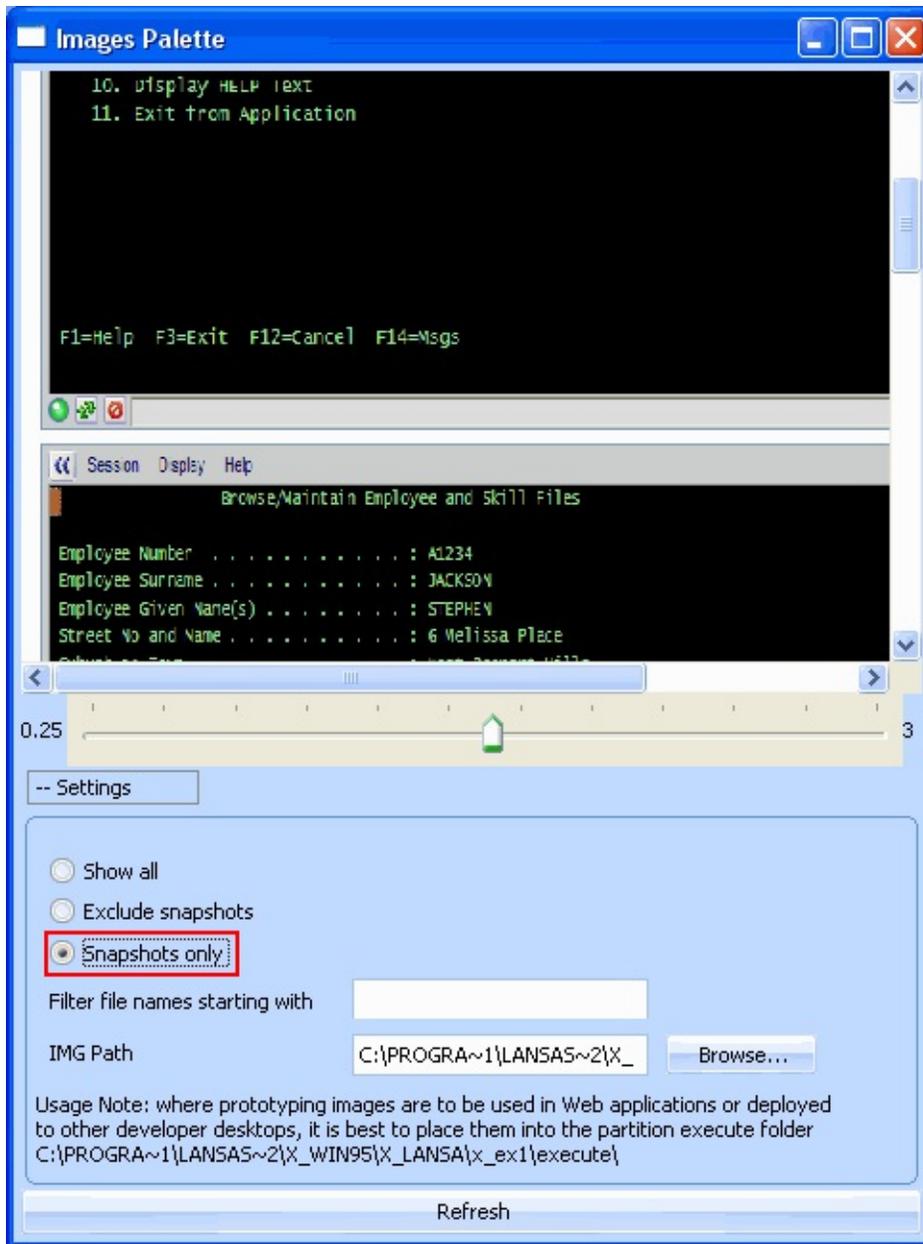
1. In the Framework, select an employee in the instance list to display the command handlers associated with employees:



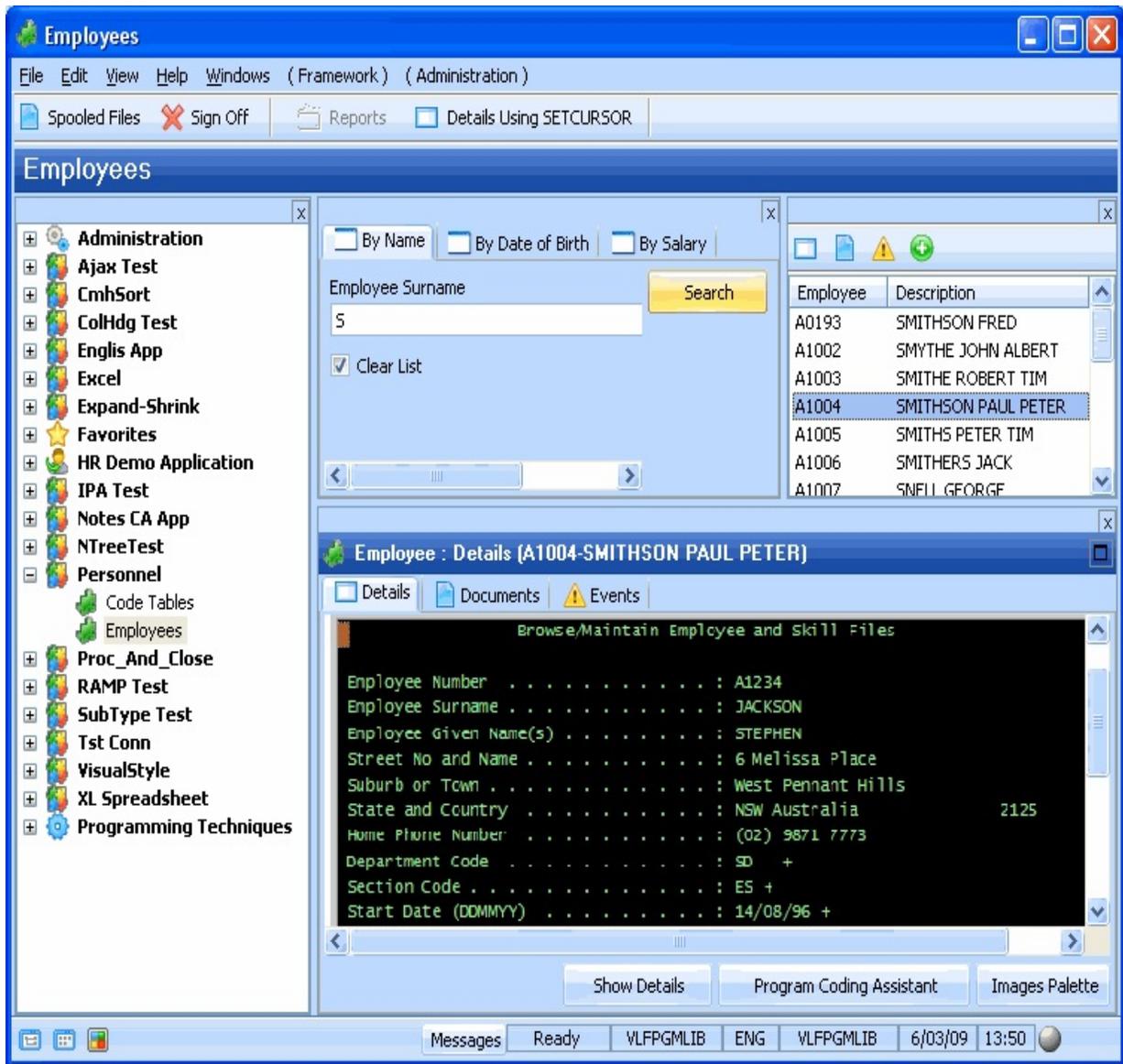
2. Like the prototype filters, the prototype command handlers are editable.
3. Select all the text in the command handler for the Details command and delete it.



4. Click on the on the Images Palette button.
5. In the Images Palette window click on the Snapshots Only radio button.



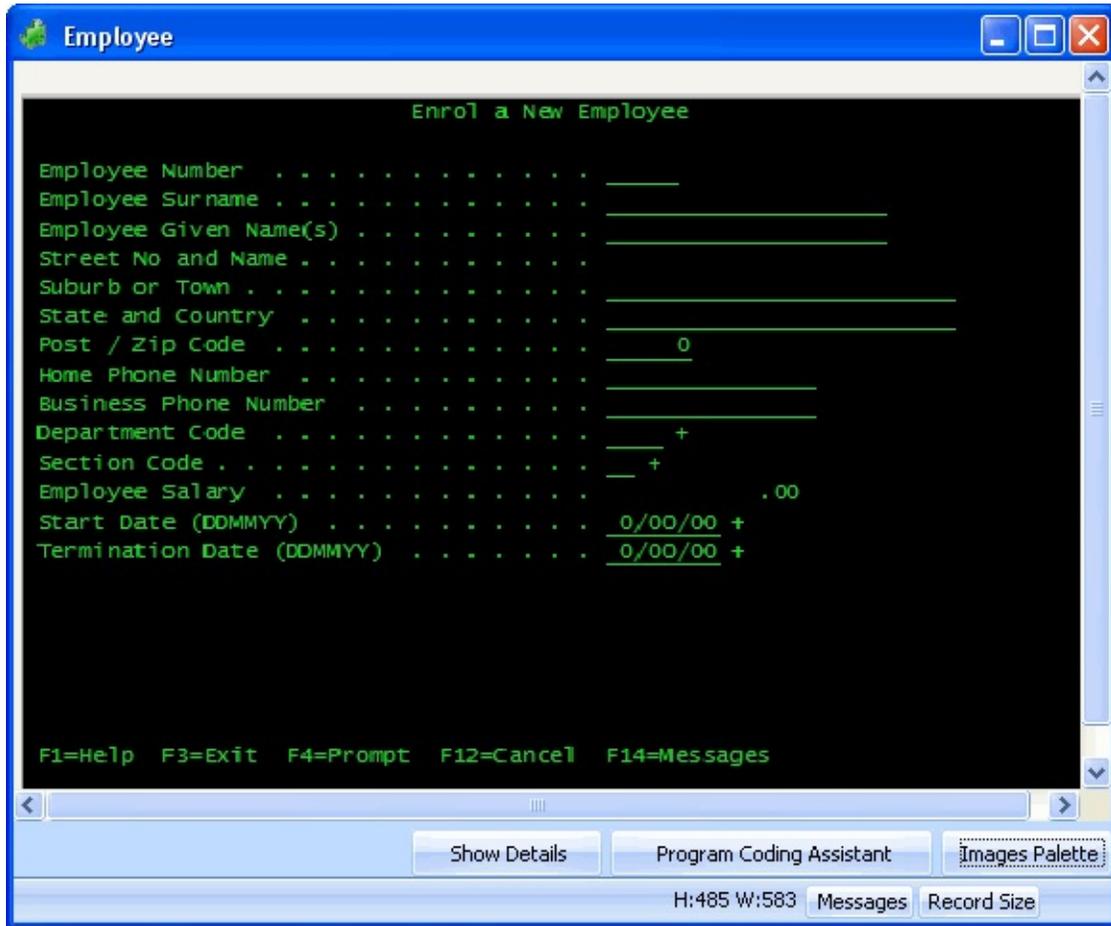
6. Locate the snapshot of the Browse/Maintain Employee and Skills Files screen and drag it to the command handler.
7. Resize the snapshot on the command handler if necessary.



Notice how much easier it is to envisage what the modernized application will look like when the command handlers show a snapshot of the screen to be snapped in.

8. Display the command handler for the New command and click the Images Palette button.
9. Locate the Enrol New Employee screen and right-click to bring up the context menu and select Copy.
10. Click on the Employee window and paste the image.

11. Resize the image if necessary.



RAMP-TS005 Step 2. Review the Prototype

When the prototype is visually complete, it is recommended you review it with the end-users to ensure the design corresponds to their expectations.

The most successful RAMP solutions have all been created on the basis of careful prototyping.

Summary

Important Observations

Showing your end-users a realistic prototype will ensure they are able to understand your design and can provide meaningful feedback.

For command handlers that will use screen wrappers or Visual LANSA components, use text and the images palette.

Snapshots are useful also after the prototyping stage. You can use them in project documentation and as a reference point if you change your screens as part of the modernization process. When you have snapped the actual screens to the Framework you can display the snapshots using the Show Snapshot button.

What You Should Know

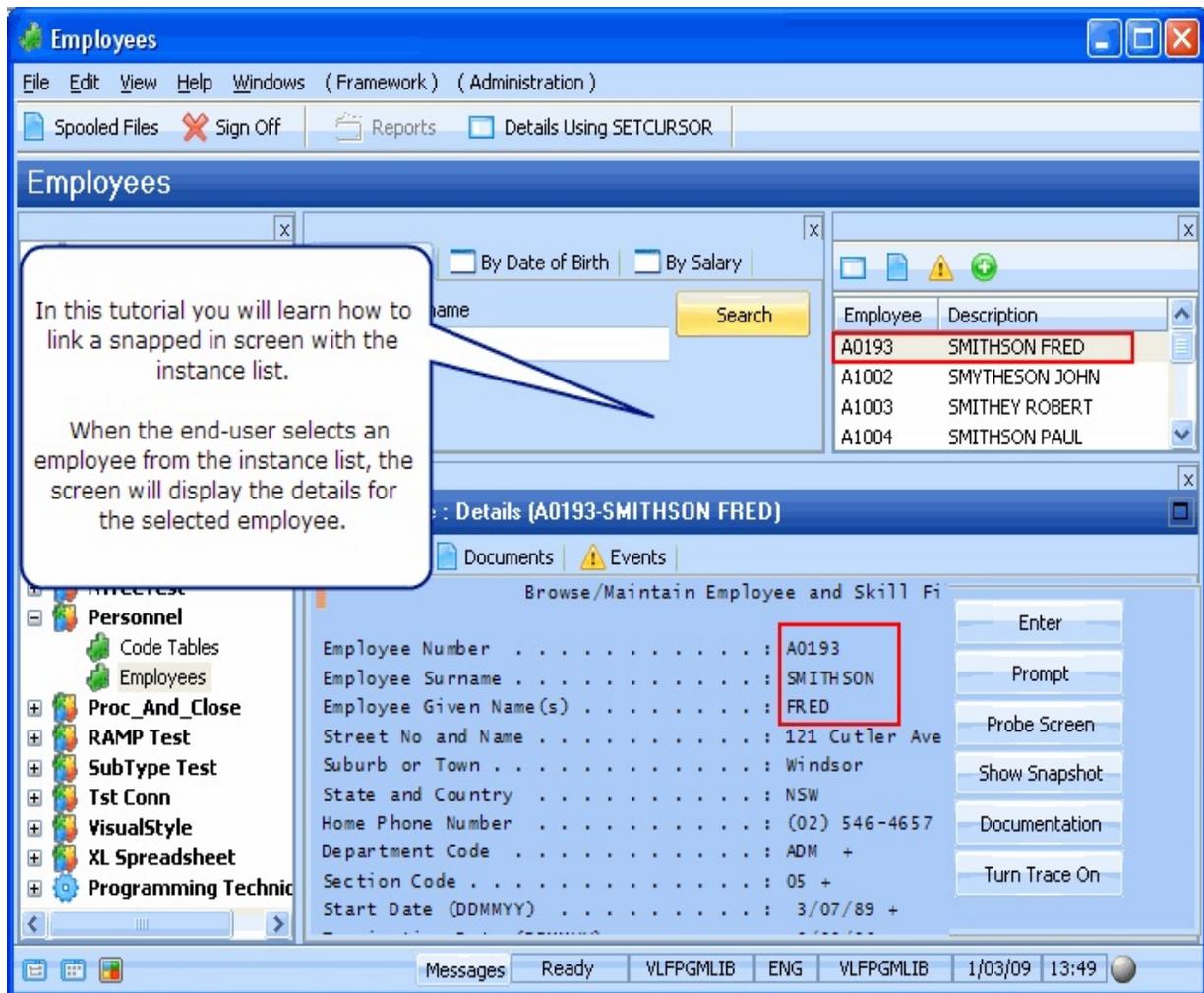
How to make your prototype visually complete.

RAMP-TS006: Snapping in a Basic Inquiry Screen

In this tutorial you will snap the Browse/Maintain Employee Details and Skills File screen to the Framework. It is a typical basic 5250 inquiry screen.

Objectives

Learn how to associate your snapped in screen with the instance list



To achieve this objective, you will complete the following steps:

[RAMP-TS006 Step 1. Snap a Basic Inquiry Screen into the Framework](#)

[RAMP-TS006 Step 2. Change the Script to Use the Current Instance List Entry](#)

[RAMP-TS006 Step 3. Disable Function Keys](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

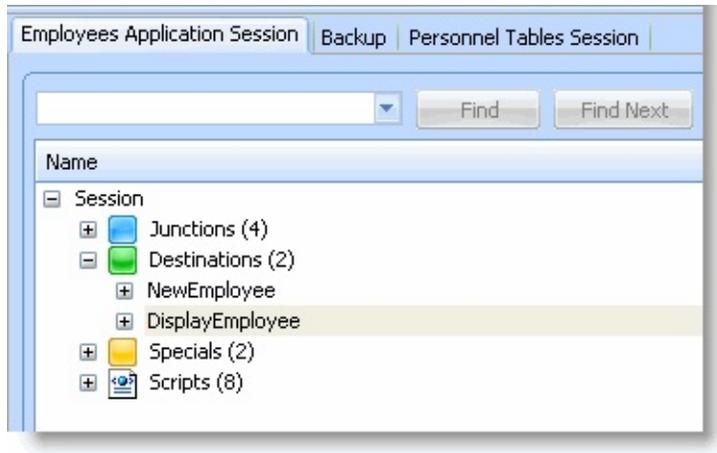
[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)[RAMP-TS005: Reviewing Design](#)

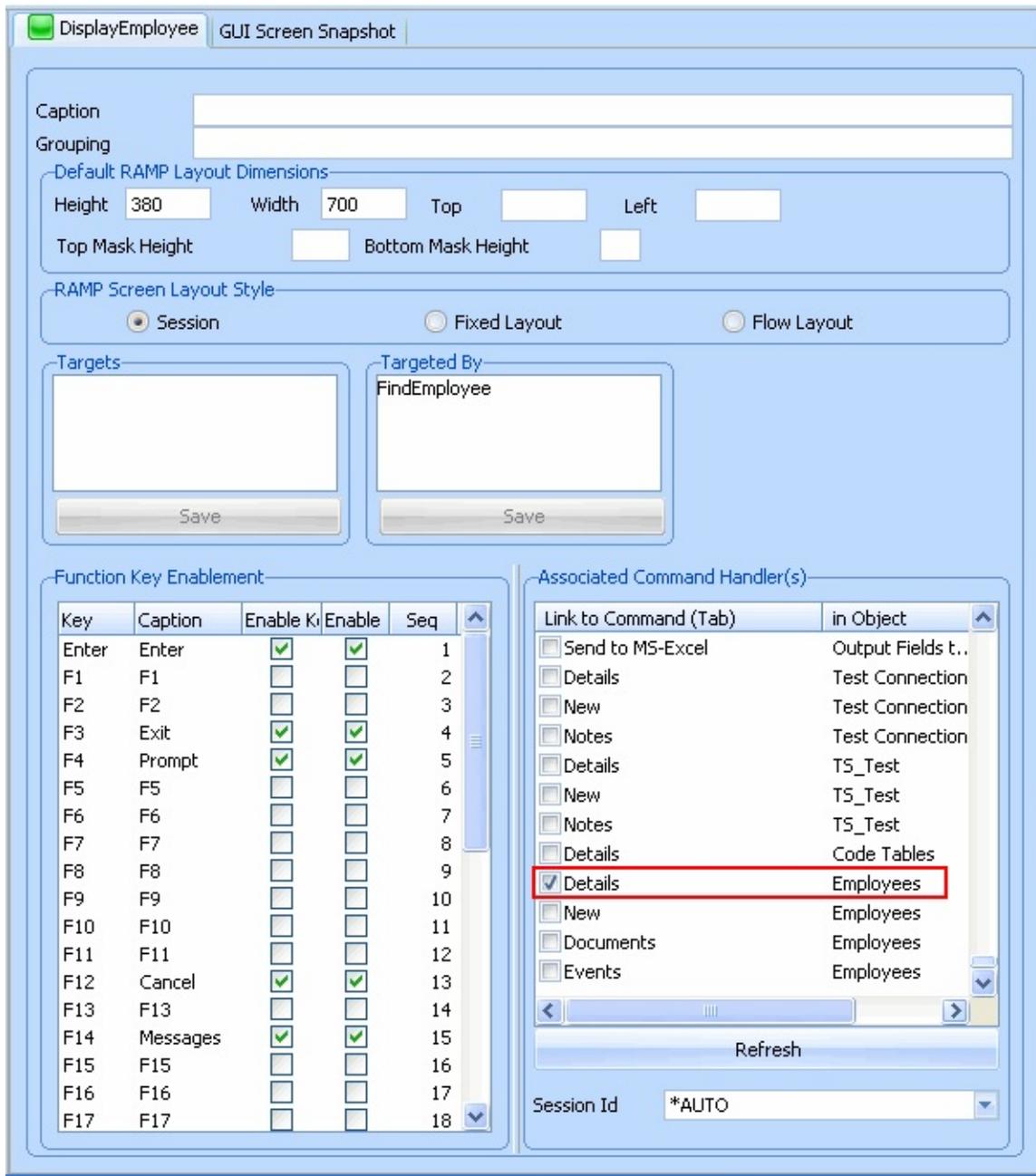
RAMP-TS006 Step 1. Snap a Basic Inquiry Screen into the Framework

In this step you will snap in the Browse/Maintain Employee Details and Skills File screen to the Framework and test it.

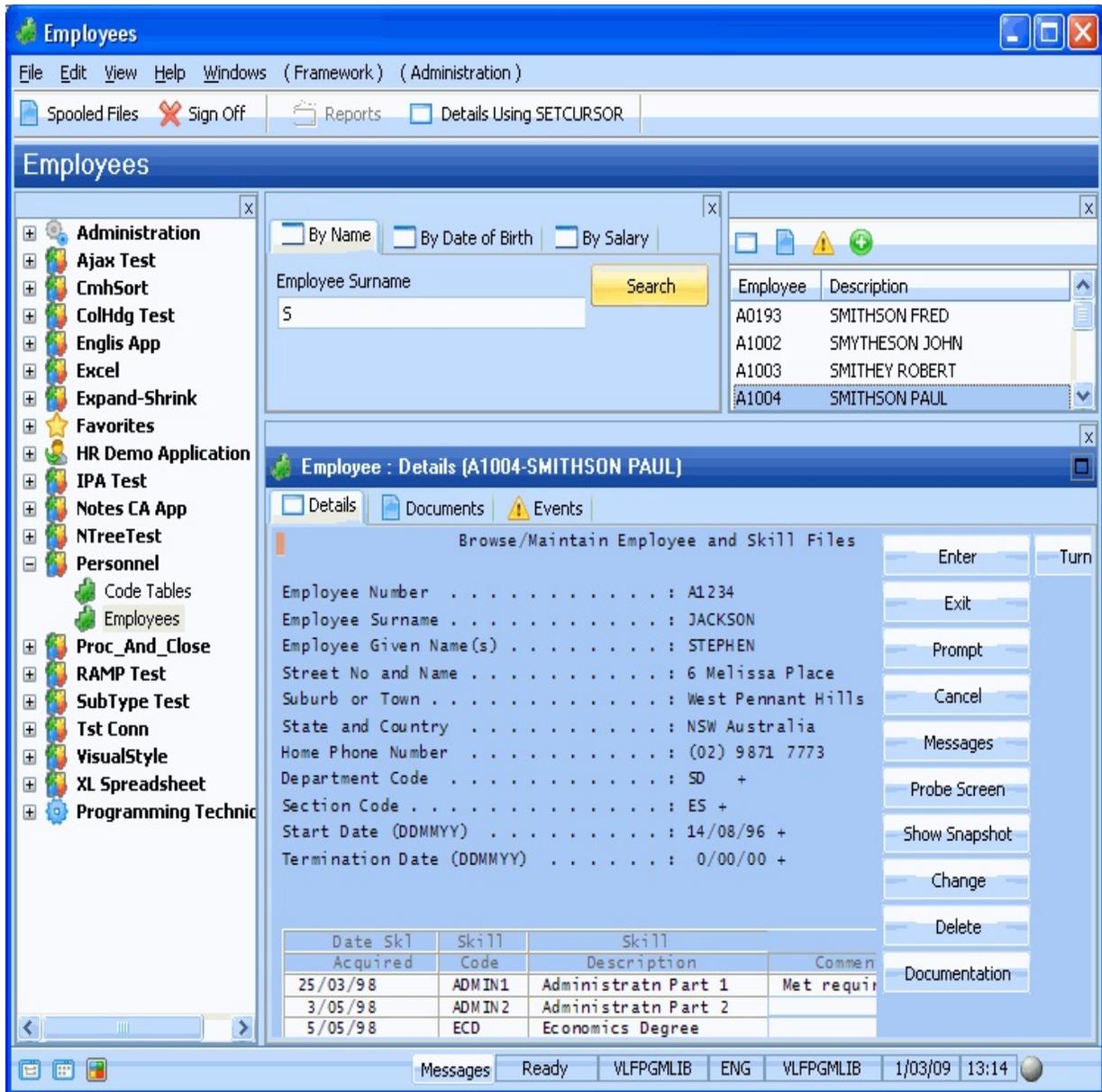
1. In the RAMP Tools window, select the DisplayEmployee destination screen in the [Screen and Script List](#).



2. In the DisplayEmployee details, snap the screen into the Details command handler of the Employees business object:



3. Do a partial save.
4. Display the Framework and select an employee from the instance list to test the command handler:



If you have correctly named and classified the screens and tracked the navigation, the Browse/Maintain Employee and Skill Files screen is now functional in the Framework.

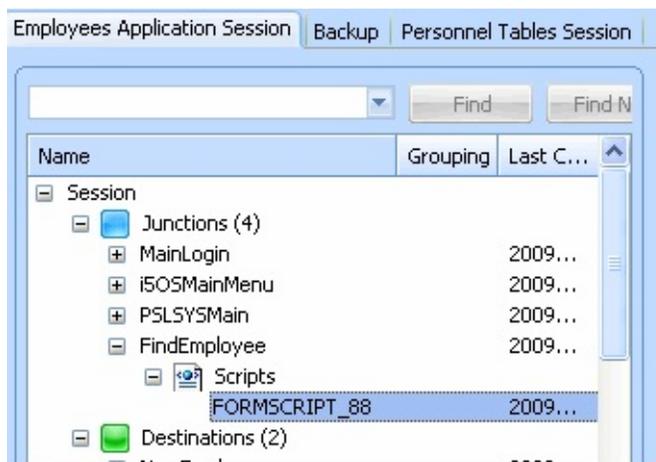
However, notice that if you select an employee in the instance list, this change is not reflected in the command handler. This is because the script which was generated to display the screen has recorded the employee number you used when tracking the screens.

RAMP-TS006 Step 2. Change the Script to Use the Current Instance List Entry

In this step you will change the script that displays the employee details by replacing the hardcoded employee number with the employee number of the currently selected instance list entry.

As you are completing this exercise you may want to refer to [Javascript Essentials](#).

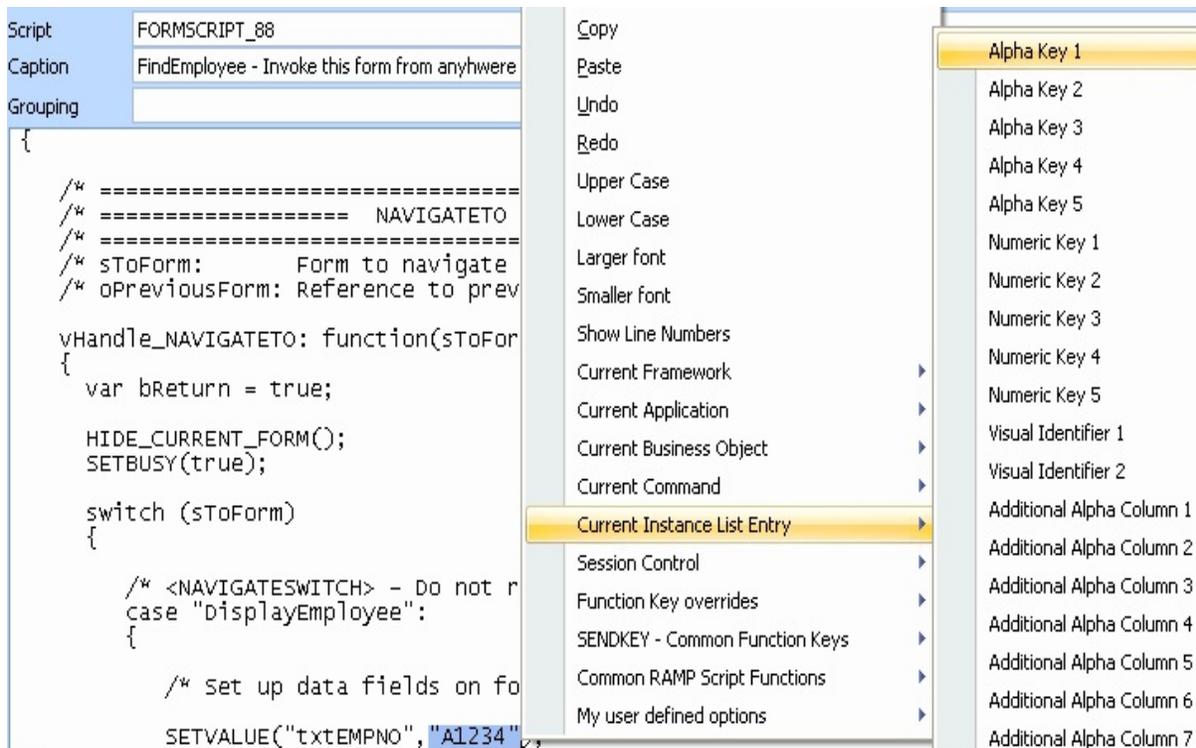
1. Select the FindEmployee junction screen in the screens and scripts list.
2. Expand it to display the script associated with it.
3. Select the script in the list to display its contents:



4. Locate this statement in the NAVIGATETO function:

```
SETVALUE("txtEmpno","A1234");
```

5. Select the Employee Number and the quotes surrounding it and right-click to bring up the context menu.
6. Select the Current Instance List Entry option in the menu.
7. Select Alpha Key 1 in the submenu:



The statement is changed to:

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

This special value will contain the identifier of the employee that the end-user has selected in the instance list.

8. Commit the changes to the script by clicking the Commit Changes button:



9. Do a partial save of the RAMP information

10. Display the Framework and select an employee in the instance list.

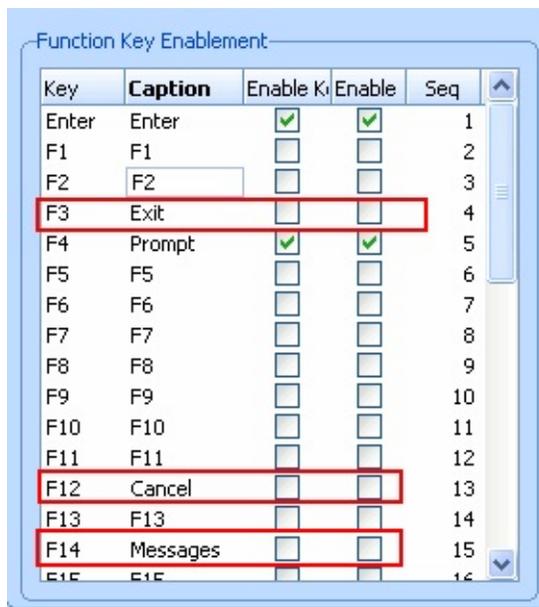
Notice that the correct details for the selected employee are now shown. If for some reason, you cannot display the employee details correctly, restart the Framework.

RAMP-TS006 Step 3. Disable Function Keys

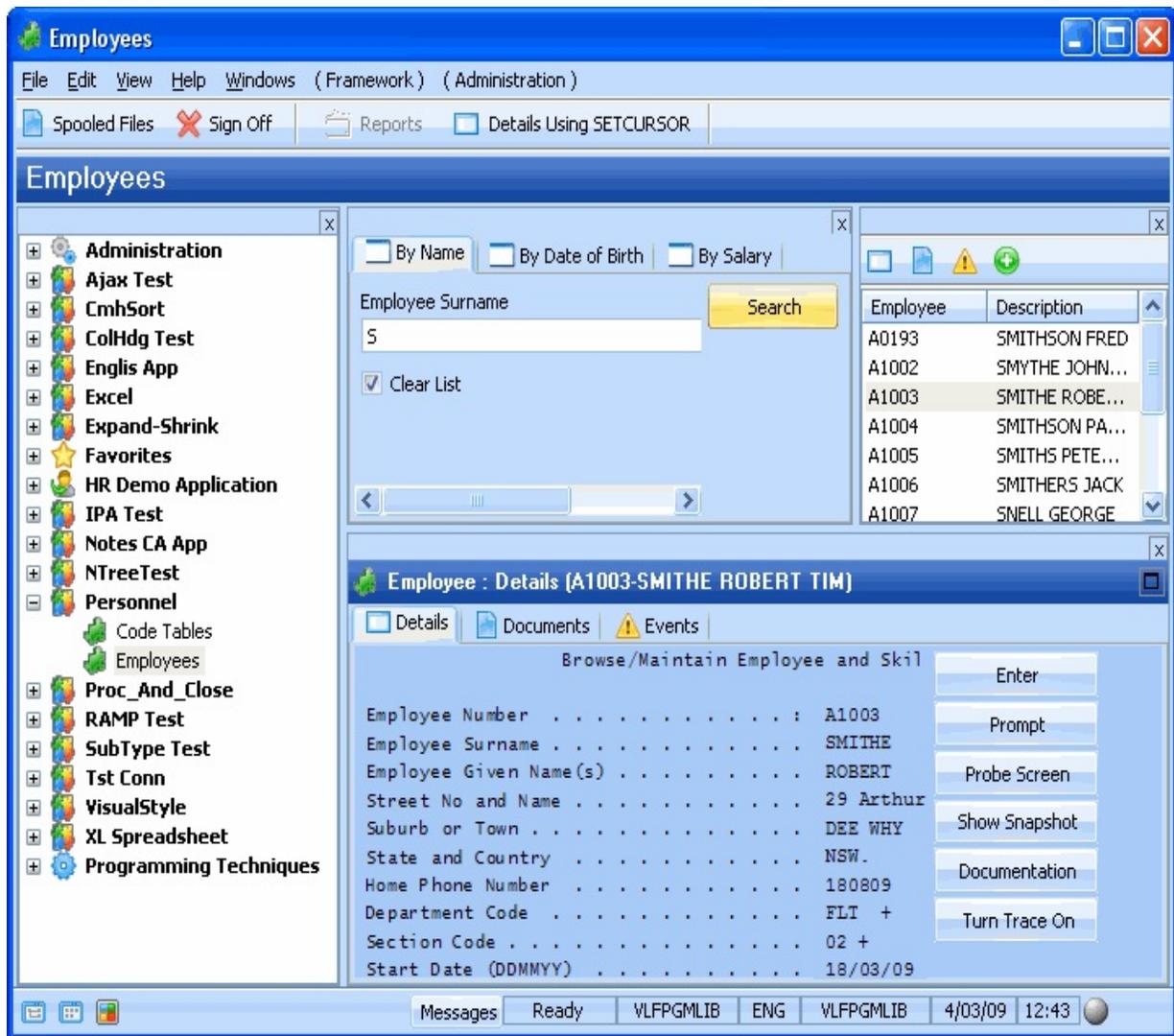
In this step you will disable the 5250 keys used for navigation because in the Framework you can move wherever you want with a few mouse clicks so the use of function keys and buttons for navigation purposes is now superfluous.

You should aim to eliminate the 5250 navigation buttons in your modernization project because it makes your application more familiar and comfortable to people who are used to the Windows interface.

1. Display the RAMP Tools window.
2. Select the DisplayEmployee screen in the [Screen and Script List](#).
3. In the Function Key Enablement list disable the Exit, Cancel and Messages keys and buttons.



4. Do a partial save of the RAMP definition.
5. Switch to the Framework and display the details of an employee to verify the buttons are no longer displayed and that the keys do not work.



The Probe Screen, Show Snapshot, Documentation and Turn Trace On buttons are RAMP design-time only buttons.

Summary

Important Observations

Automatically generated scripts contain the hard coded number of the value you chose when tracking navigation. You need to make a change to the script to make the screen respond to the currently selected entry in the instance List.

Use the [Scripting Pop-up Menu](#) to change your scripts.

What You Should Know

How to make the screen interact with the instance list.

RAMP-TS007: Snapping in a Data Entry Function

In this tutorial you will snap the Enrol a New Employee screen to the Framework. It is a typical 5250 data entry screen.

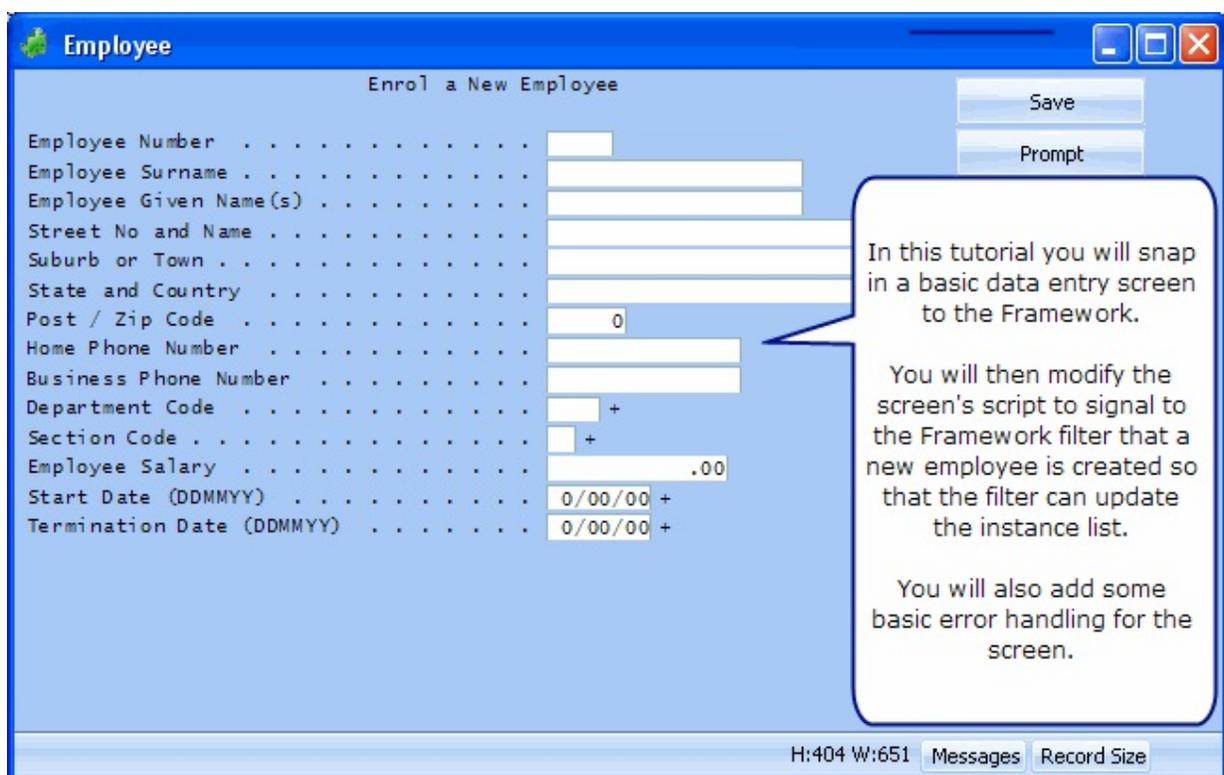
Objectives

Learn how to snap a data entry function into the Framework.

Understand the different functions in a screen's script.

Learn how the screen can signal events to the Framework filter.

Learn how to create basic error handling for a screen.



To achieve this objective, you will complete the following steps:

- [RAMP-TS007 Step 1. Snap a Basic Data Entry Screen into the Framework](#)
- [RAMP-TS007 Step 2. Change the Script to Signal the New Employee Number](#)
- [RAMP-TS007 Step 3. Add Error Handling](#)
- [RAMP-TS007 Step 4. Change the Script to Update the Instance List](#)
- [RAMP-TS007 Step 5. Disable Function Keys](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

RAMP-TS007 Step 1. Snap a Basic Data Entry Screen into the Framework

In this step you will snap in the Browse/Maintain Employee Details and Skills File screen to the Framework and test it.

1. In the RAMP Tools window, select the NewEmployee destination screen in the [Screen and Script List](#).
2. In the NewEmployee details snap the screen into the New command handler of the Employees business object.

The screenshot shows the 'NewEmployee' configuration window in RAMP Tools. The window title is 'NewEmployee GUI Screen Snapshot'. It contains several sections for configuring the screen's appearance and behavior.

Default RAMP Layout Dimensions:

- Height: 380
- Width: 700
- Top: [empty]
- Left: [empty]
- Top Mask Height: [empty]
- Bottom Mask Height: [empty]

RAMP Screen Layout Style:

- Session
- Fixed Layout
- Flow Layout

Targets: [empty list box]

Targeted By: PLSYSMain

Function Key Enablement:

Key	Caption	Enable K	Enable	Seq
Enter	Enter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2
F2	F2	<input type="checkbox"/>	<input type="checkbox"/>	3
F3	Exit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4
F4	Prompt	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5
F5	F5	<input type="checkbox"/>	<input type="checkbox"/>	6
F6	F6	<input type="checkbox"/>	<input type="checkbox"/>	7
F7	F7	<input type="checkbox"/>	<input type="checkbox"/>	8
F8	F8	<input type="checkbox"/>	<input type="checkbox"/>	9
F9	F9	<input type="checkbox"/>	<input type="checkbox"/>	10
F10	F10	<input type="checkbox"/>	<input type="checkbox"/>	11
F11	F11	<input type="checkbox"/>	<input type="checkbox"/>	12
F12	Cancel	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	13

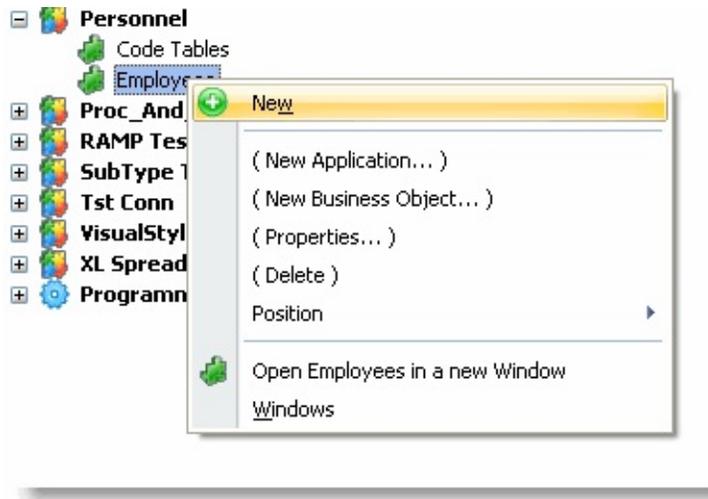
Associated Command Handler(s):

Link to Command (Tab)	in Object	U...
<input type="checkbox"/> Notes	Test Connection	T..
<input type="checkbox"/> Details	TS_Test	T..
<input type="checkbox"/> New	TS_Test	T..
<input type="checkbox"/> Notes	TS_Test	T..
<input type="checkbox"/> Details	Code Tables	C..
<input type="checkbox"/> Details	Employees	E..
<input checked="" type="checkbox"/> New	Employees	E..
<input type="checkbox"/> Documents	Employees	E..
<input type="checkbox"/> Events	Employees	E..

Refresh

Session Id: *AUTO

3. Do a partial save.
4. Display the Framework and select the Employees business object in the navigation pane, use the filter to fill the instance list and then right-click the Employees business object or an individual employee in the instance list to display the context menu.
5. Select New.



The Enrol New Employee screen is displayed in the Framework.

Employee

Enrol a New Employee

Employee Number

Employee Surname

Employee Given Name(s)

Street No and Name

Suburb or Town

State and Country

Post / Zip Code 0

Home Phone Number

Business Phone Number

Department Code +

Section Code +

Employee Salary00

Start Date (DDMMYY) 0/00/00 +

Termination Date (DDMMYY) 0/00/00 +

Enter

Exit

Prompt

Cancel

Messages

Probe Screen

Change

Show Snapshot

Delete

Documentation

Turn Trace On

H:404 W:651 Messages Record Size

6. Enter the details for a new employee and press Enter to save.

7. Search for the newly added employee using the By Name filter.

In the next step you will change the New Employee screen to update the instance list automatically.

RAMP-TS007 Step 2. Change the Script to Signal the New Employee Number

In this step you will add code to your screen script signal the new employee number when a new employee is created.

1. In the RAMP Tools window, locate the script for the NewEmployee screen.



2. Locate the BUTTONCLICK function, then the statement declaring bReturn variable.
3. Add this statement to declare an oEmp object which will be used to pass the employee number:

```
var oEmp = new Object();
```

Your code should look like this:

```
vHandle_BUTTONCLICK: function(sButton)
{
    var bReturn = true;
    var oEmp = new Object();
    if (HANDLE_PROMPT()) return(bReturn); /* If the focus element is au
```

4. Then locate the statement handling the pressing of the Enter key (KeyEnter).
5. Add this statement before the SENDKEY statement to retrieve the value of the employee number of the new employee to the oEmp object:

```
oEmp.empno = GETVALUE("txtEmpno");
```

6. Change the SENDKEY function to pass the employee number payload:

```
SENDKEY(KeyEnter, oEmp);
```

Your code will look like this:

```
vHandle_BUTTONCLICK: function(sButton)
{
    var bReturn = true;
    var oEmp = new Object();
    if (HANDLE_PROMPT()) return(bReturn); /* If the focus element is au
    /* <BUTTONCLICK /> - Do not remove or alter this line */
        /* Handle function keys and buttons */
        switch (sButton)
        {
            case KeyEnter:
                oEmp.empno = GETVALUE("txtEmpNO");
                SENDKEY(KeyEnter, oEmp);
                break;
            case KeyF3:
                SENDKEY(KeyF3);
                break;
        }
    }
}
```

7. Commit the changes.

You will use the employee number payload in the following steps.

RAMP-TS007 Step 3. Add Error Handling

In this step you will add code to your script to issue a message if the new employee insertion was successful.

After any attempted insert (whether successful or not), the NewEmployee screen is redisplayed. Therefore you can place the error handling code in the ARRIVE script of the screen. You will then add code to close the New Employee window if a new employee has successfully been added.

For the error handling you will need a Javascript function to trim the employee number.

1. Add this function code to the end of your script, just before the SYSINFO:

```
/* =====  
/* ===== uTrim ===== */  
/* =====  
/* sStringToTrim: The string to be trimmed left and right */  
  
uTrim : function (sStringToTrim) {  
  
return sStringToTrim.replace(/^s+|s+$/g,"");  
  
} ,
```

Your code will look like this:

```

        /* </NAVIGATESWITCH> - Do not remove or alter this line */
    return(bReturn);
},

```

```

/* ===== */
/* ===== uTrim ===== */
/* ===== */
/* sstringToTrim: The string to be trimmed left and right */

uTrim : function (sstringToTrim) {
return sstringToTrim.replace(/^s+|\s+$/g, "");
} ,

```

```

//<SYSINFO>
vName      : "NewEmployee",
vGUID      : "B2561D0512924940A595417625362687",

```

You will only want to execute the error handling code when the NewEmployee screen is redisplayed, so you will first need to check what the previous screen was:

2. Add this statement after the declaration of the bReturn variable in the vHandle_ARRIVE function:

```

if (oPreviousForm.vName == "NewEmployee" )
{
}

```

To differentiate between a successful insert and a validation error add code to check whether the employee number has been set to blank by the operation. If it is not blank, close the window:

3. Add this code to the if statement:

```

/* Get the currently showing EMPNO */
var strCurrEmpno = GETVALUE("txtEmpno");
if ((this.uTrim(strCurrEmpno) == "") && (this.uTrim(oPayload.empno) !
{
    /* Insert was successful */
    /* Issue a message */
    MESSAGE("Employee ", oPayload.empno , " created");
    AVCLOSEFORM();
}

```

```
}
```

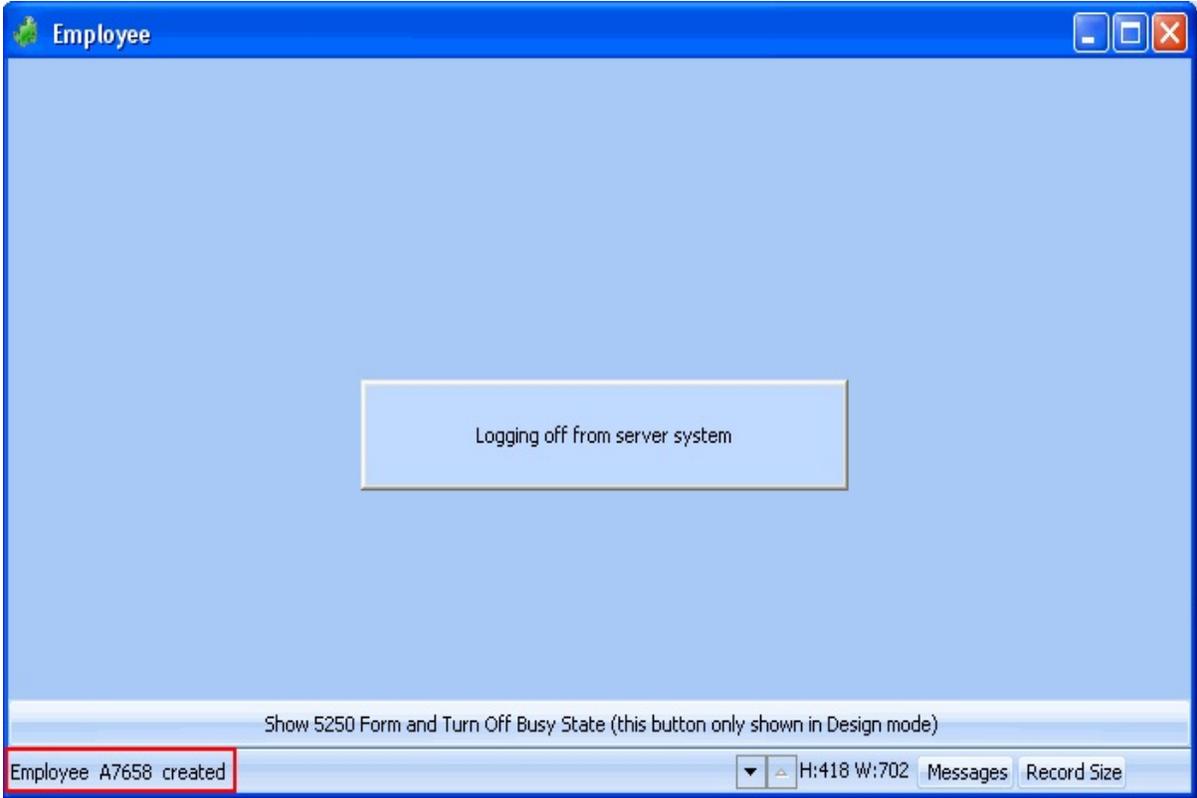
Your code will look like this:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;

    if (oPreviousForm.vName == "NewEmployee" )
    {
        /* Get the currently showing EMPNO */
        var strCurrEmpno = GETVALUE("txtEmpno");
        if ((this.uTrim(strCurrEmpno) == "") && (this.uTrim(oPayload.empno) != "")) )
        {
            /* Insert was sucessful */
            /* Issue a message */
            MESSAGE("Employee ", oPayload.empno ," created");
            AVCLOSEFORM();
        }
    }

    SHOW_CURRENT_FORM(true); /* show the form in the framework and show VLF buttons */
}
```

4. Click on Commit Changes and then do a partial save.
5. In the Framework add a new employee. Notice that after you have successfully added an employee, the message is displayed and the window is closed.



RAMP-TS007 Step 4. Change the Script to Update the Instance List

In this step you will add code to your screen script to update the instance list when a new employee is created.

1. Add this statement after the MESSAGE function in the error handling code to send a signal to your filter to update the instance list after an employee has been successfully created:

```
AVSIGNALEVENT("Add_List_Entry","BUSINESSOBJECT",  
oPayload.empno)
```

Your code will look like this:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)  
{  
  var bReturn = true;  
  
  if (oPreviousForm.vName == "NewEmployee" )  
  {  
  
    /* Get the currently showing EMPNO */  
    var strCurrEmpno = GETVALUE("txtEmpno");  
    if ((this.uTrim(strCurrEmpno) == "") && (this.uTrim(oPayload.empno) != ""))  
    {  
      /* Insert was successful */  
      /* Issue a message */  
      MESSAGE("Employee ", oPayload.empno , " created");  
      AVSIGNALEVENT("Add_List_Entry","BUSINESSOBJECT", oPayload.empno)  
      AVCLOSEFORM();  
    }  
  }  
  
  SHOW_CURRENT_FORM(true); /* Show the form in the framework and show VLF buttons */  
}
```

When you created your filter in [RAMP-TS003 Step 1. Creating Your Real By Name Filter](#), you specified that the filter should contain code to listen for signals to update the instance list, so the code to listen for this signal already exists in the filter.

2. Save the code changes by clicking on the Commit Changes button.
3. Do a partial save of the RAMP definition.
4. Display the Framework.

5. Use the filter to locate all employees whose surname starts with S.
6. Enter the details for a new employee whose name starts with S.

The screenshot shows a window titled "Employee" with a subtitle "Enrol a New Employee". The form contains the following fields and values:

Employee Number	A8345
Employee Surname	SALINGER
Employee Given Name(s)	BETTY
Street No and Name	58 Surrey St
Suburb or Town	North Sydney
State and Country	NSW Australia
Post / Zip Code	2001
Home Phone Number	988340343
Business Phone Number	3433098767
Department Code	ADM +
Section Code	01 +
Employee Salary	100.00
Start Date (DDMMYY)	010101 +
Termination Date (DDMMYY)	020202 +

On the right side of the form, there is a vertical list of buttons: Enter, Exit, Prompt, Cancel, Messages, Probe Screen, Change, Show Snapshot, Delete, Documentation, and Turn Trace On. The "Enter" button is highlighted in yellow.

At the bottom right of the window, there is a status bar with the text "H:418 W:702 Messages Record Size".

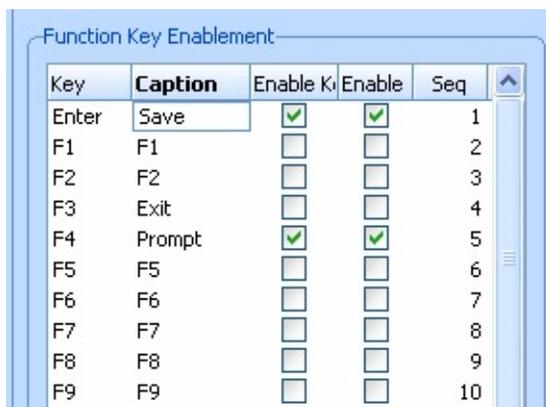
7. Click Enter. Notice that the newly created employee is displayed in the instance list.

Employee	Description
A0193	SMITHSON FRED
A1002	SMYTHE JOHN ALBERT
A1003	SMITHE ROBERT TIM
A1004	SMITHSON PAUL PETER
A1005	SMITHS PETER TIM
A1006	SMITHERS JACK
A1007	SNELL GEORGE
A1008	SNEDDON ALLAN
A1009	SNASHALL DAMIAN
A2367	SALINGER TIM
A3145	SNOW BRIDGET
A3876	SAUNDERS JAMES
A4709	SNOWDON EDWARD
A8345	SALINGER BETTY
A8966	SNOWDEN PETER
A9800	SPRESSER DF

RAMP-TS007 Step 5. Disable Function Keys

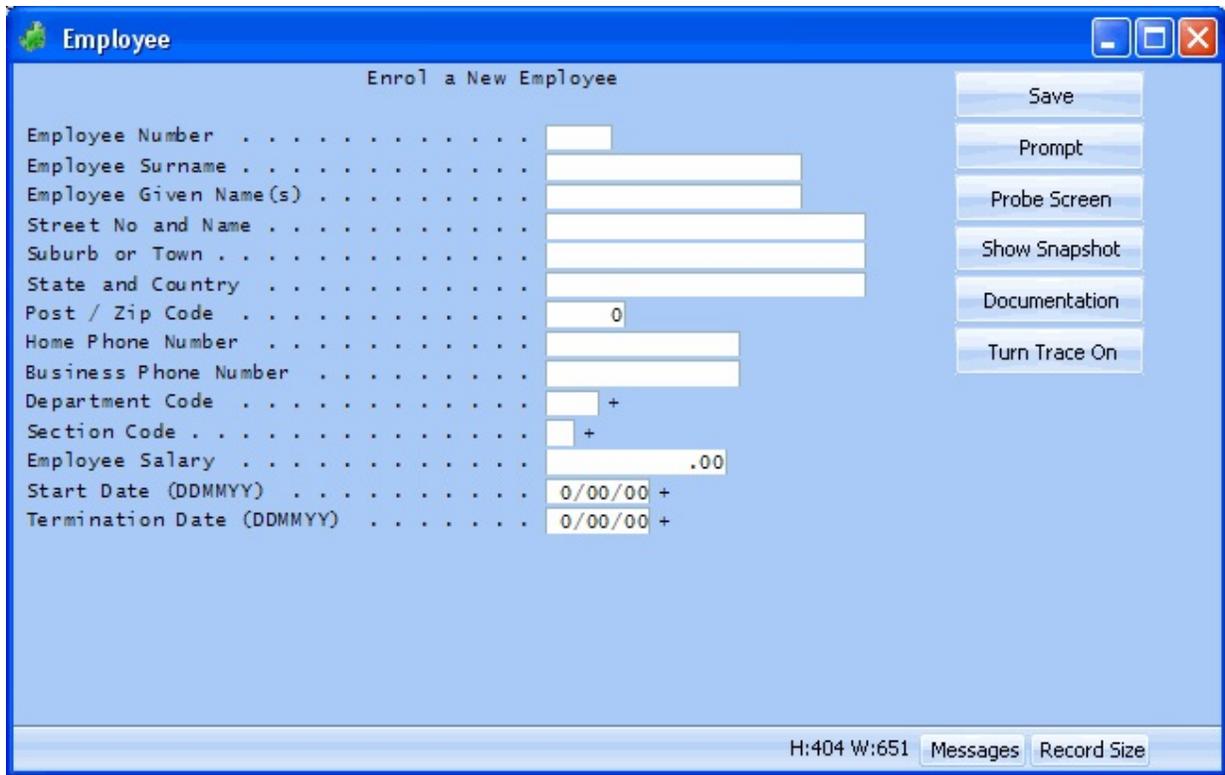
In this step you will disable function keys which are not required in the modernized application. You will also hide the corresponding buttons.

1. In the RAMP Tools window, select NewEmployee in the screens and script list to display its details.
2. In the Function Key Enablement list disable all keys/buttons except Enter and Prompt.
3. Change the caption of Enter to Save.



Key	Caption	Enable Key	Enable	Seq
Enter	Save	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2
F2	F2	<input type="checkbox"/>	<input type="checkbox"/>	3
F3	Exit	<input type="checkbox"/>	<input type="checkbox"/>	4
F4	Prompt	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5
F5	F5	<input type="checkbox"/>	<input type="checkbox"/>	6
F6	F6	<input type="checkbox"/>	<input type="checkbox"/>	7
F7	F7	<input type="checkbox"/>	<input type="checkbox"/>	8
F8	F8	<input type="checkbox"/>	<input type="checkbox"/>	9
F9	F9	<input type="checkbox"/>	<input type="checkbox"/>	10

4. Do a partial save of the RAMP definition.
5. Display the Framework, click to add a new employee and check that the correct buttons are shown.



The four bottommost buttons are only shown when running the Framework in Design mode.

Summary

Important Observations

The SENDKEY function has an optional Payload parameter you can use to pass information

To signal events to the Framework filter, use the AvSignalEvent function.

What You Should Know

How to snap a data entry function into the Framework.

What are the different functions in a screen's script

How the screen can signal events to the Framework filter.

How to create basic error handling for a screen.

RAMP-TS008: Changing Inquiry Screen to Update Mode

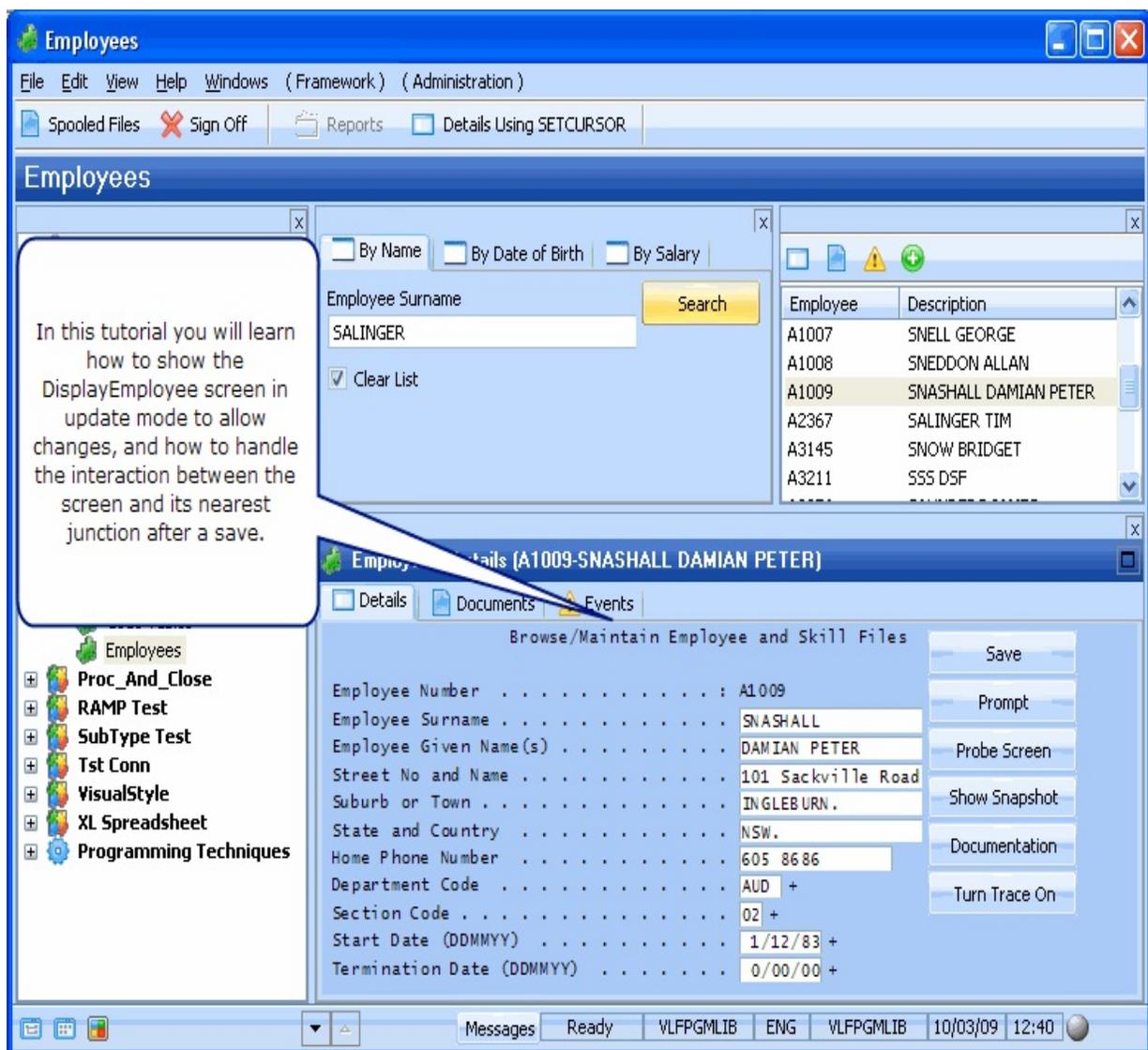
In this tutorial you will change the EmployeeDetails screen to update mode to allow changes.

Objectives

Learn how to change an inquiry screen to an update screen.

Learn how to handle navigation between the update screen and its nearest junction.

Understand more about the structure of scripts.



To achieve this objective, you will complete the following steps:

[RAMP-TS008 Step 1. Make Display Employee Screen Input Capable](#)

[RAMP-TS008 Step 2. Redisplay DisplayEmployee After Save](#)

[RAMP-TS008 Step 3. Change Button Caption](#)

-

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

RAMP-TS008 Step 1. Make Display Employee Screen Input Capable

In this step you will display the DisplayEmployee screen in update mode.

The tutorial [RAMP-TS006: Snapping in a Basic Inquiry Screen](#) showed how to snap in a basic read-only inquiry screen to the Framework just to demonstrate some basic steps, but in real-life modernized applications you would as a rule display the screens as input capable.

In [RAMP-TS004 Step 1. Name the Screens](#) you named the txtSurname field on the DisplayEmployee screen. The field is only displayed when the screen is in change mode so you can use it to determine and set the mode of the screen.

1. In the RAMP Tools window select the DisplayEmployee screen in the [Screen and Script List](#).
2. Expand it to display the script associated with the screen.
3. In the vHandle_ARRIVE function add this code after the definition of bReturn to see if the txtSurname field exists, and if it does not, to set the screen to change mode:

```
if (CHECK_FIELD_EXISTS("txtSurname"))
{

}

else
{
SENDKEY(KeyF21);
}
```

Your code should look like this:

```

i
/* ===== */
/* ===== vHandle_ARRIVE ===== */
/* ===== */
/* Handle arrival at this Destination */
/* oPayload: The payload supplied by the event initiator */
/* oPreviousForm: Reference to previous object Form*/

vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
  var bReturn = true;

  if (CHECK_FIELD_EXISTS("txtsurname"))
  {
    SHOW_CURRENT_FORM(true); /* Show the form in the framework and show VLF buttons */
    HIDE_5250_BUTTONS();    /* Hide any 5250 style buttons displayed */
    GET_FORM_MESSAGE(22);   /* Extract messages and hide the message line */
    SETBUSY(false);        /* Last thing done - turn off the busy state */
  }
  else
  {
    SENDKEY(KeyF21);
  }

  /* <ARRIVE /> - Do not remove or alter this line */

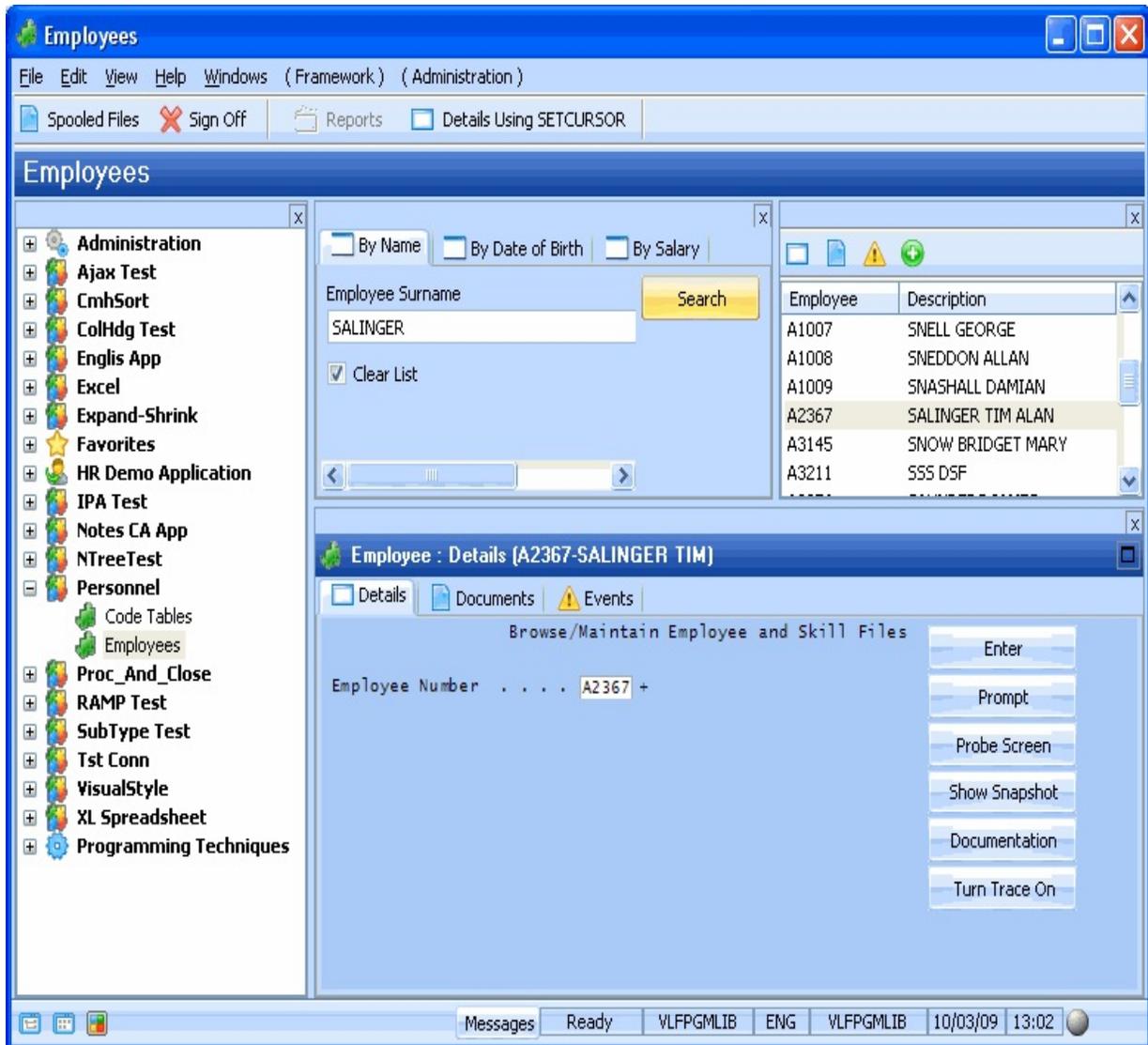
  return(bReturn);
},

/* ===== */
/* ===== BUTTONCLICK ===== */
/* ===== */

```

4. Commit the changes.
5. Do a partial save of the RAMP definition.
6. Display the Framework.
7. Select an employee in the instance list. Notice that the Browse/Maintain Employee and Skill Files screen is now displayed in change mode.
8. Make a change to the name of the employee.
9. Press Enter.

Notice that after the save the FindEmployee screen is shown.



This is typical 5250 behavior which needs to be changed to fit the Framework navigation model. In the next step you will make the FindEmployee script to redisplay the Browse/Maintain Employee and Skill Files screen after employee details have been saved.

RAMP-TS008 Step 2. Redisplay DisplayEmployee After Save

In this step you will change the script for the FindEmployee screen to navigate back to the Browse/Maintain Employee and Skill Files screen after the details of an employee have been saved.

1. Locate the vHandle_ARRIVE function in the script for FindEmployee.
2. Under the declaration `var bReturn = true;` create a new object:

```
var oEmp = new Object();
```

3. Below the statement `/* <ARRIVE /> - Do not remove or alter this line */` add code to check if the previous screen is DisplayEmployee:

```
if (oPreviousForm.vName == "DisplayEmployee")  
{  
}
```

4. In the if statement signal to the Framework that that the filter needs to update the instance list with the new employee:

```
oEmp.empno = GETVALUE("txtEmpno");  
AVSIGNALEVENT("Update_List_Entry","BUSINESSOBJECT", oEmp.
```

5. And add this code to navigate back to the DisplayEmployee screen:

```
NAVIGATE_TO_SCREEN("DisplayEmployee");
```

Your code will now look like this:

```

/* ===== */
/* ===== vHandle_ARRIVE ===== */
/* ===== */
/* Handle arrival at this JUNCTION */
/* oPayload: The payload supplied by the event initiator */
/* oPreviousForm: Reference to previous object Form */

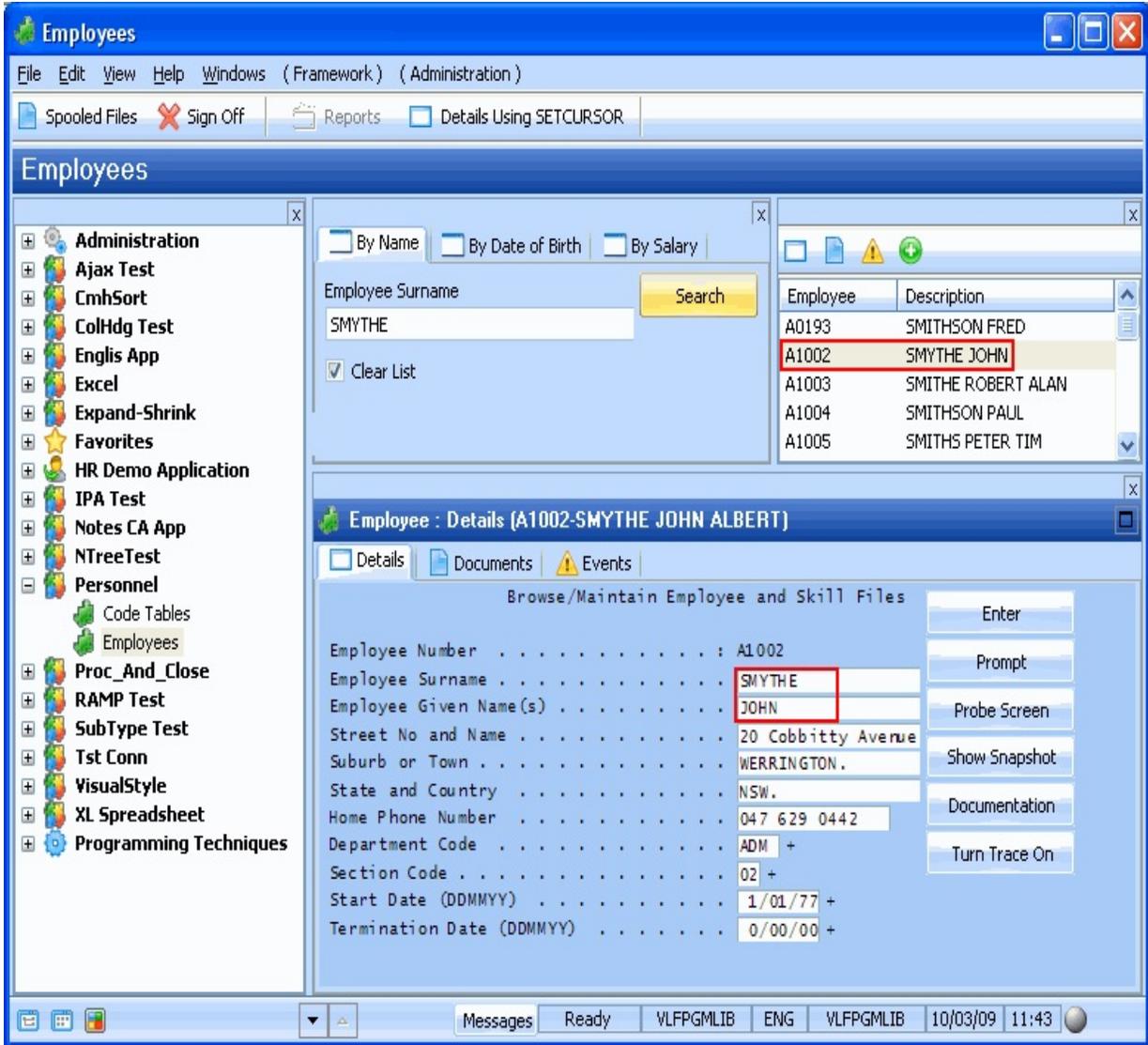
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;
    var oEmp = new Object();

    /* <ARRIVE /> - Do not remove or alter this line */

    if (oPreviousForm.vName == "DisplayEmployee")
    {
        oEmp.empno = GETVALUE("txtEMPNO");
        AVSIGNAL EVENT("Update_List_Entry", "BUSINESSOBJECT", oEmp.empno);
        NAVIGATE_TO_SCREEN("DisplayEmployee");
    }
    return(bReturn);
},

```

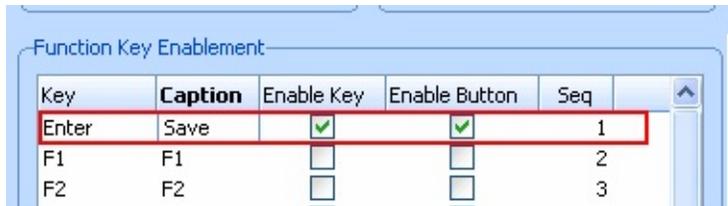
6. Commit changes and then do a partial save.
7. In the Framework update the details of an employee. Notice that the DisplayEmployee screen is now redisplayed and the instance list reflects the changes to the employee details:



RAMP-TS008 Step 3. Change Button Caption

In this step you will change the caption of the Enter button to Save.

1. In the RAMP Tools window, select the DisplayEmployee screen.
2. In the Function Key Enablement list, select the caption of the Enter button and change it Save.



Key	Caption	Enable Key	Enable Button	Seq
Enter	Save	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2
F2	F2	<input type="checkbox"/>	<input type="checkbox"/>	3

3. Hide and disable the Change and Delete buttons.
4. Do a partial save of the RAMP definition.
5. Switch to the Framework display the details of an employee and verify the function key caption.

Summary

Important Observations

You can create an input capable screen simply by tracking navigation, RAMP will automatically update the associated script.

After a successful save, the previous junction screen is displayed. You can add code in the `vHandle_Arrive` function in the script associated with the junction screen.

What You Should Know

How to snap in a screen in update mode.

How to handle navigation between the destination screen and its nearest junction after a save.

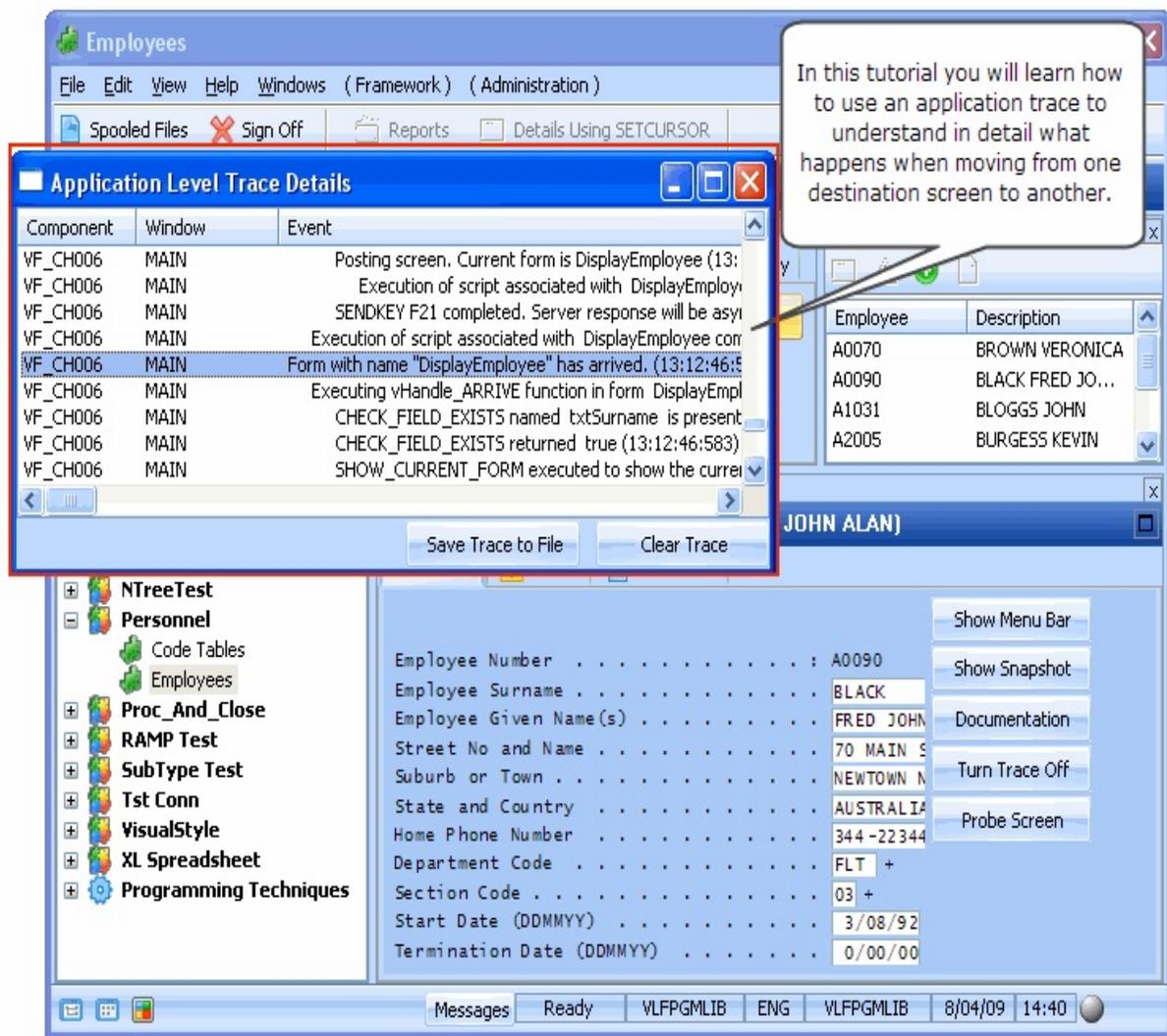
RAMP-TS009: Tracing Navigation

In this tutorial you will use application tracing to understand what happens when you move from a destination screen to another.

Objectives

Learn how to use application tracing

Understand the functions and commands used in navigation scripts



To achieve this objective, you will complete the following steps:

[RAMP-TS009 Step 1. Starting the Trace and Redisplaying the Destination](#)

Screen

[RAMP-TS009 Step 2. Examining the Trace](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

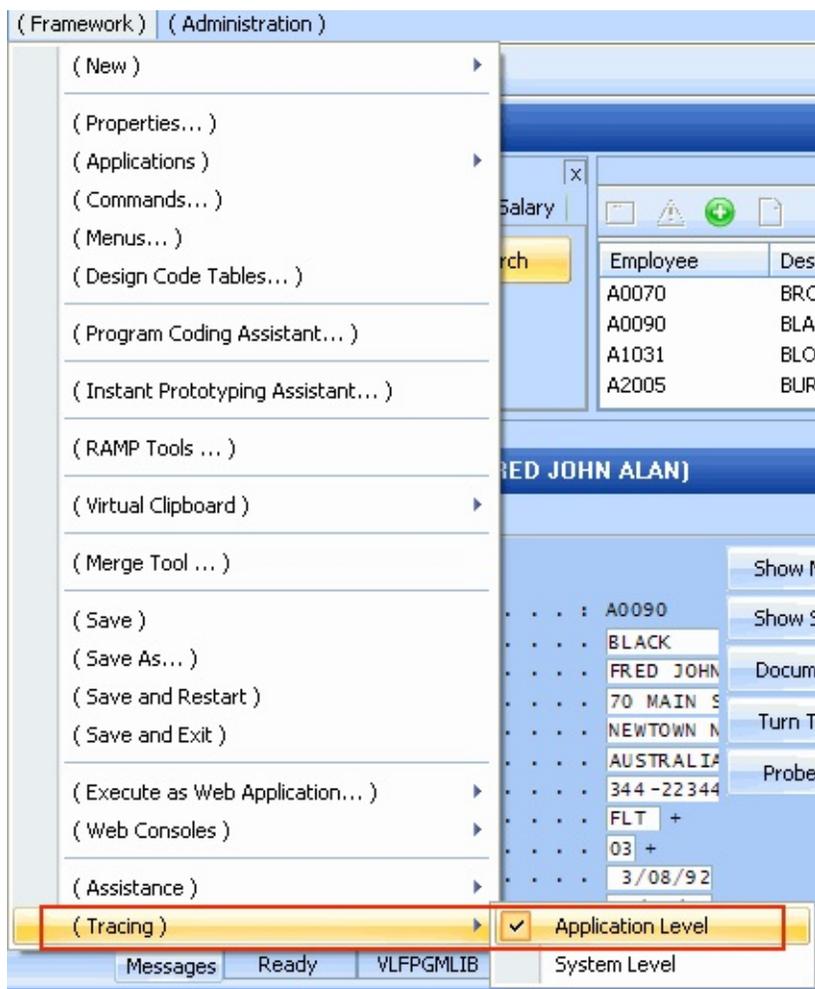
[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

RAMP-TS009 Step 1. Starting the Trace and Redisplaying the Destination Screen

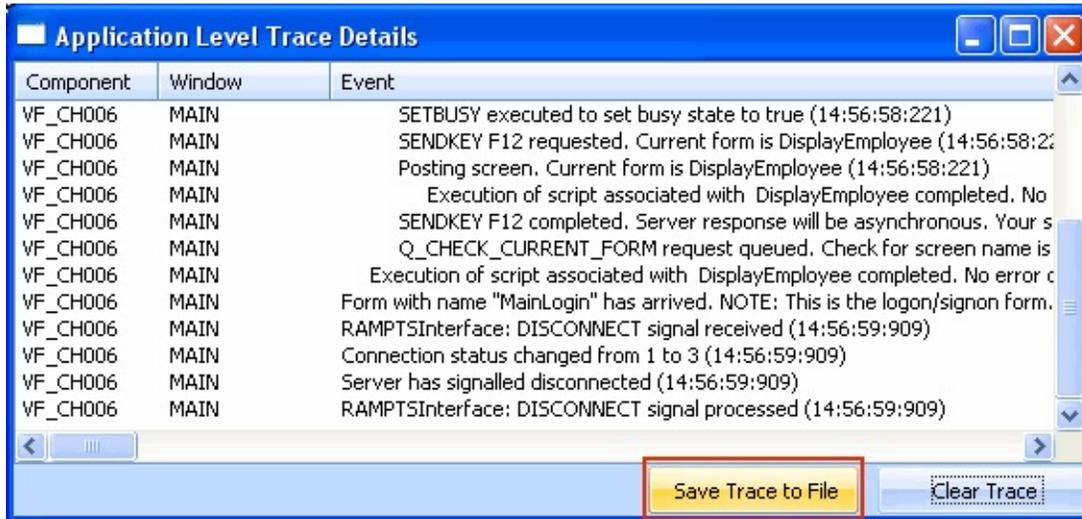
In this step you will start an application trace and then redisplay the destination screen in order to understand what happens when a destination screen is displayed.

1. Select an Employee in the instance list of the Personnel application.
2. When the details for the employee are displayed, start an application trace using the Framework menu:



3. Now select another employee in the instance list. Notice that the trace details are shown in the window.
4. Click on the Save Trace to File button to save the trace details to a file and

then open it in Notepad.



RAMP-TS009 Step 2. Examining the Trace

In this step examine the trace:

Command Handler - uExecute executed
GetCurrentInstance returns VisualID1(A0090) VisualID2(BLACK FRED JOHN ALAN) AN
Script manager handling request STARTNAVIGATION. The current form is DisplayEmployee
Preparing a navigation plan to get to screen DisplayEmployee (15:2:11:921)
Initial navigation will request that the currently displayed destination screen
Preparation of final navigation plan to get to screen DisplayEmployee has been
Executing vHandle_NAVIGATETO function in form DisplayEmployee (15:2:11:921)
HIDE_CURRENT_FORM executed to hide the current 5250 screen (15:2:11:921)
SETBUSY executed to set busy state to true (15:2:11:921)
SENDKEY F12 requested. Current form is DisplayEmployee (15:2:11:921)
Posting screen. Current form is DisplayEmployee (15:2:11:921)
Execution of script associated with DisplayEmployee completed.
SENDKEY F12 completed. Server response will be asynchronous. Your script should end now. Current form is DisplayEmployee (15:2:11:937)
Q_CHECK_CURRENT_FORM request queued. Check for screen name is PLSYSMain Current form is DisplayEmployee (15:2:11:937)
Execution of script associated with DisplayEmployee completed. No error detected. (15:2:11:937)

When the Details command handler for the selected employee is executed, RAMP starts building a navigation plan to display the requested 5250 screen.

The vHandle_NAVIGATETO function in the current screen is executed to exit to the PLSYSMain screen.

Form with name "PLSYSMain" has arrived. (15:2:12:234)
Executing vHandle_ARRIVE function in form PLSYSMain (15:2:12:234)
Execution of script associated with PLSYSMain completed. No error detected.
CHECK_CURRENT_FORM check passed. Current form is PLSYSMain (15:2:12:234)
Preparing a navigation plan to get to screen DisplayEmployee (15:2:12:234)
An optimized pre-existing navigation plan from PLSYSMain to DisplayEmployee was found.
The navigation plan from screen PLSYSMain to DisplayEmployee is (15:2:12:234)
-> Initially, request that screen PLSYSMain navigates to screen FindEmployee
-> Finally, request that screen FindEmployee navigates to screen DisplayEmployee
End of navigation plan (15:2:12:234)
Executing vHandle_NAVIGATETO function in form PLSYSMain (15:2:12:234)
HIDE_CURRENT_FORM executed to hide the current 5250 screen (15:2:12:234)
SETBUSY executed to set busy state to true (15:2:12:234)
SETVALUE of txtOption index = 1 to value 3 (type = string) .Current form is PLSYSMain (15:2:12:234)
SETVALUE of txtOption completed. Current form is PLSYSMain (15:2:12:234)
SENDKEY Enter requested. Current form is PLSYSMain (15:2:12:234)
Posting screen. Current form is PLSYSMain (15:2:12:234)
Execution of script associated with PLSYSMain completed. No error detected. (15:2:12:234)
SENDKEY Enter completed. Server response will be asynchronous. Your script should end now. Current form is PLSYSMain (15:2:12:249)
Q_CHECK_CURRENT_FORM request queued. Check for screen name is FindEmployee Current form is PLSYSMain (15:2:12:249)
Execution of script associated with PLSYSMain completed. No error detected. (15:2:12:249)

The PLSYSMain screen arrives. First its vHandle_ARRIVE function is executed and a navigation plan to reach the DisplayEmployee screen is created.

The vHandle_NAVIGATETO function is then executed to display the FindEmployee screen.

Form with name "FindEmployee" has arrived. (15:2:12:624)
Executing vHandle_ARRIVE function in form FindEmployee (15:2:12:624)
Execution of script associated with FindEmployee completed. No error detected.
CHECK_CURRENT_FORM check passed. Current form is FindEmployee (15:2:12:624)
Executing vHandle_NAVIGATETO function in form FindEmployee (15:2:12:624)
HIDE_CURRENT_FORM executed to hide the current 5250 screen (15:2:12:624)
SETBUSY executed to set busy state to true (15:2:12:624)
SETVALUE of txtEMPNO index = 0 to value A0090 (type = string) .Current form is FindEmployee (15:2:12:640)
SETVALUE of txtEMPNO completed. Current form is FindEmployee (15:2:12:640)
SENDKEY Enter requested. Current form is FindEmployee (15:2:12:640)
Posting screen. Current form is FindEmployee (15:2:12:640)
Execution of script associated with FindEmployee completed. No error detected. (15:2:12:640)
SENDKEY Enter completed. Server response will be asynchronous. Your script should end now. Current form is FindEmployee (15:2:12:640)
Q_CHECK_CURRENT_FORM request queued. Check for screen name is DisplayEmployee Current form is FindEmployee (15:2:12:655)
Execution of script associated with FindEmployee completed. No error detected. (15:2:12:655)

The FindEmployee screen arrives and the vHandle_NAVIGATETO function sets the employee number and then navigates to the DisplayEmployee screen.

Form with name "DisplayEmployee" has arrived. (15:2:13:155)
Executing vHandle_ARRIVE function in form DisplayEmployee (15:2:13:155)
CHECK_FIELD_EXISTS named txtSurname is present on the current screen (15:2:13:155)
CHECK_FIELD_EXISTS returned false (15:2:13:155)
SENDKEY F21 requested. Current form is DisplayEmployee (15:2:13:155)
Posting screen. Current form is DisplayEmployee (15:2:13:155)
Execution of script associated with DisplayEmployee completed. No error detected.
SENDKEY F21 completed. Server response will be asynchronous. Your script should end now. Current form is DisplayEmployee (15:2:13:17)
Execution of script associated with DisplayEmployee completed. No error detected.

When the DisplayEmployee screen arrives, its vHandle_ARRIVE script checks to see if the txtSurname field is present on the screen. It is not, because the screen is in display mode, so the script sends function key 21 to set the screen in edit mode.

Form with name "DisplayEmployee" has arrived. (15:2:13:718)
Executing vHandle_ARRIVE function in form DisplayEmployee (15:2:13:718)
CHECK_FIELD_EXISTS named txtSurname is present on the current screen (15:2:13:718)
CHECK_FIELD_EXISTS returned true (15:2:13:718)
SHOW_CURRENT_FORM executed to show the current 5250 screen area and any as Form named DisplayEmployee is defined and classified to RAMP so function key
Form named DisplayEmployee has defined function key control. The vFKC array
Enabled as buttons : (15:2:13:718)
Enabled as keys : F2 (15:2:13:718)
HIDE_5250_BUTTONS executed. (15:2:13:718)
Message ADDED; MORE RECORDS = false (15:2:13:718)
SETBUSY executed to set busy state to false (15:2:13:718)
Execution of script associated with DisplayEmployee completed. No error detected.
CHECK_CURRENT_FORM check passed. Current form is DisplayEmployee (15:2:13:718)

The DisplayEmployee screen arrives again in edit mode. The vHandle_Arrive script verifies the txtSurname field is present and then shows the screen and sets the buttons and keys for the screen.

The navigation is now complete.

Summary

Important Observations

You can use tracing to learn to understand how RAMP navigates from one screen to another.

If you encounter a problem, start by using a tracing to resolve it. See [Debug and Diagnostics](#).

The shipped Java Script function TRACE() allows you to add your own trace statements. See [Tracing](#).

You might sometimes find that the easiest and quickest way to debug a problem is to put up a message box. See [Using ALERT_MESSAGE in Your Scripts](#).

What You Should Know

How to trace your application.

RAMP-TS010: Using Special Field Handlers

You can easily provide advanced prompting in your 5250 RAMP screens by associating simple Visual LANSA forms with fields. In this tutorial you will add a special field handler to the Department Code field to let the user choose the code from a list of departments.

Objectives

Learn the basics of using special field handling.

Learn how to add value to your 5250 screens

-

The screenshot shows a 5250 RAMP screen titled "Employees". The main window displays the details for employee "A1031-BLOGGS JOHN". The "Department Code" field is highlighted with a red box, and a dropdown menu is open, showing a list of departments including "ADMINISTRATION", "PURCHASING", "ACCOUNTING", "GROUP ACCOUNTS", "INFORMATION SERVICES", "DEVELOPMENT", "ADMINISTRATION", "PURCHASING", "ACCOUNTING", "DEPARTMENTO LEGAL", and "MANAGEMENT INFORMATION". A callout box points to the dropdown menu with the text: "In this tutorial you will add a special field handler to the Department Code field to show a list of departments the end-user can choose from."

Employees

File Edit View Help Windows (Framework) (Administration)

Spooled Files Sign Off Reports Details Using SETCURSOR

Administration
Ajax Test
CmhSort
ColHdg Test

By Name By Date of Birth By Salary

Employee Surname Search

Employee Description
A0070 BROWN VERONICA

Details [A1031-BLOGGS JOHN]

Events Documents

Save
Prompt
Probe Screen
Show Snapshot
Documentation

Number : A1031
Surname : BLOGGS
Employee Given Name(s) : JOHN
Street No and Name : 3 Woodbury Road
Suburb or Town : Winston Hills
State and Country : NSW Australia
Home Phone Number : (02) 9668 9235

Department Code : INF +
Section Code : EI +
Start Date (DDMMYY) : 18/05/
Termination Date (DDMMYY) : 0/00/

Date Acquired	Skill Code	Skill Description
10/12/95	ADMIN1	Administratn Part 1
1/03/98	ADMIN2	Administratn Part 2
30/06/96	ADVPGM	Advanced Programming

Messages Ready VLFPGMLIB E

To achieve this objective, you will complete the following steps:

[RAMP-TS010 Step 1. Naming the Field](#)

[RAMP-TS010 Step 2. Associating the Field with the Handler](#)

[RAMP-TS010 Step 3. Test the Special Field Handler](#)

-
-

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

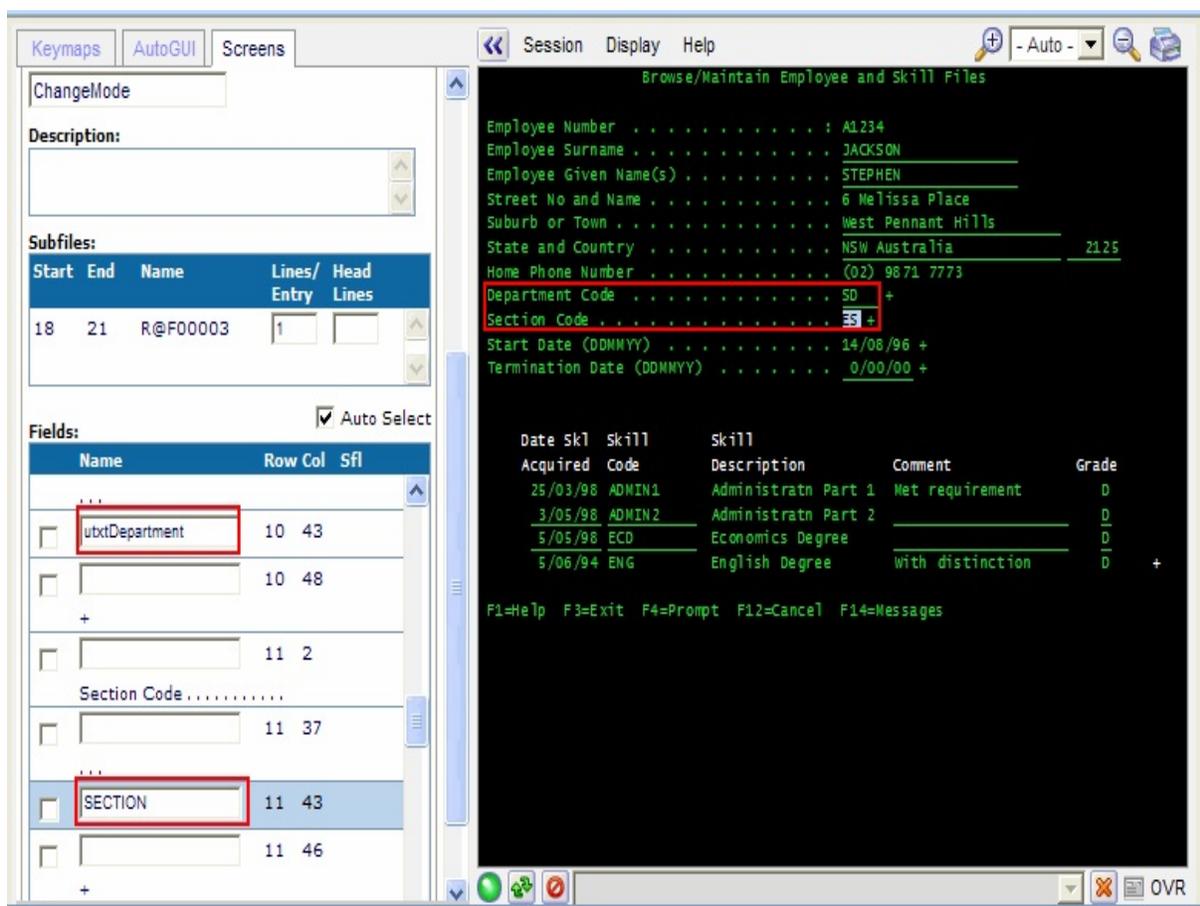
[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

[RAMP-TS009: Tracing Navigation](#)

RAMP-TS010 Step 1. Naming the Field

In this step you will name the Department Code field on the DisplayEmployee screen so that you can add special field handling to it.

1. In the RAMP Tools window start a RAMP-TS 5250 session.
2. Navigate to the DisplayEmployee screen and put it in Change mode.
3. Display the Screen naming area, locate the field showing the Department Code and name it utxtDepartment.
4. Also name the Section Code field SECTION.

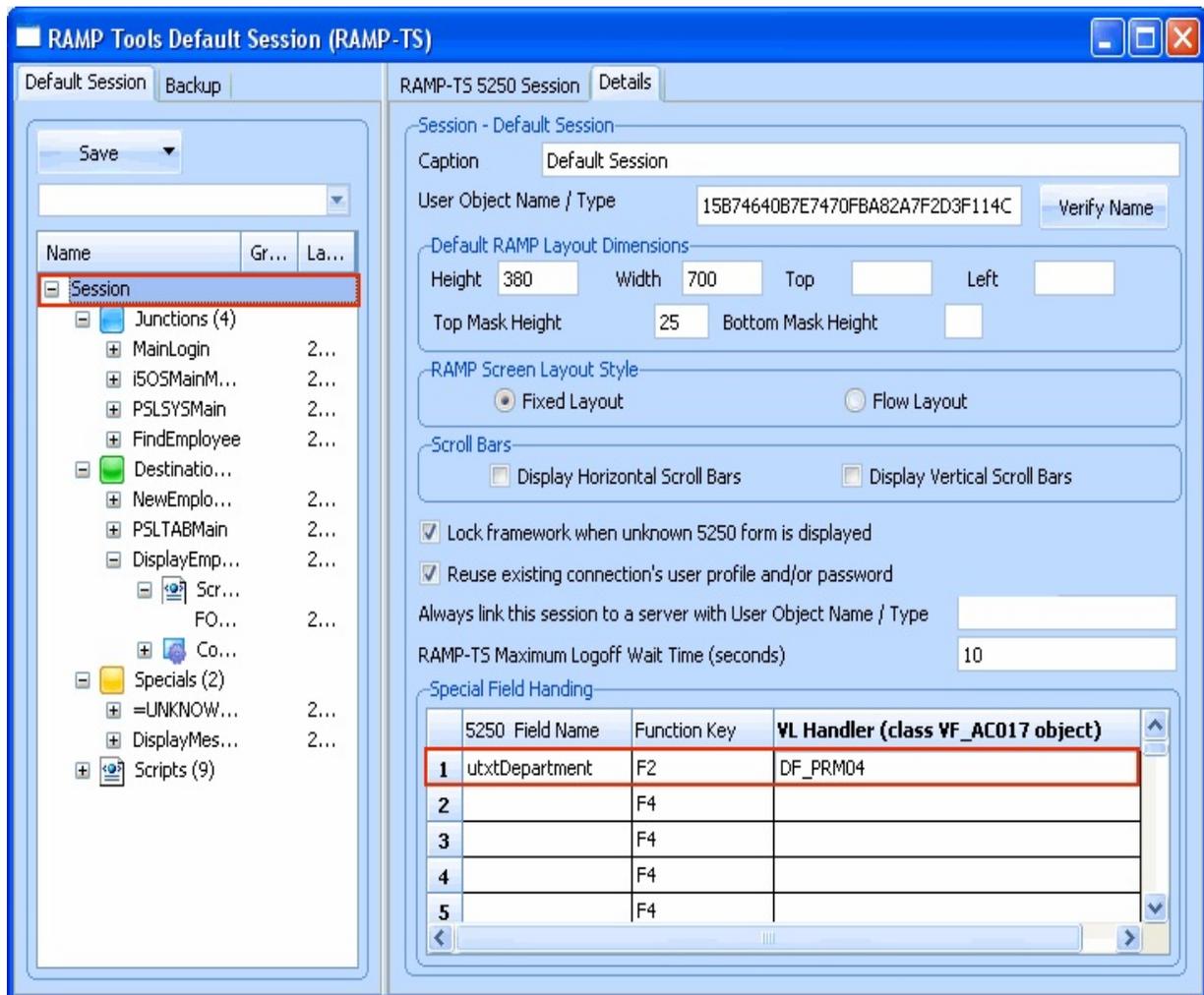


5. Click on the Save button to save the screen definition.

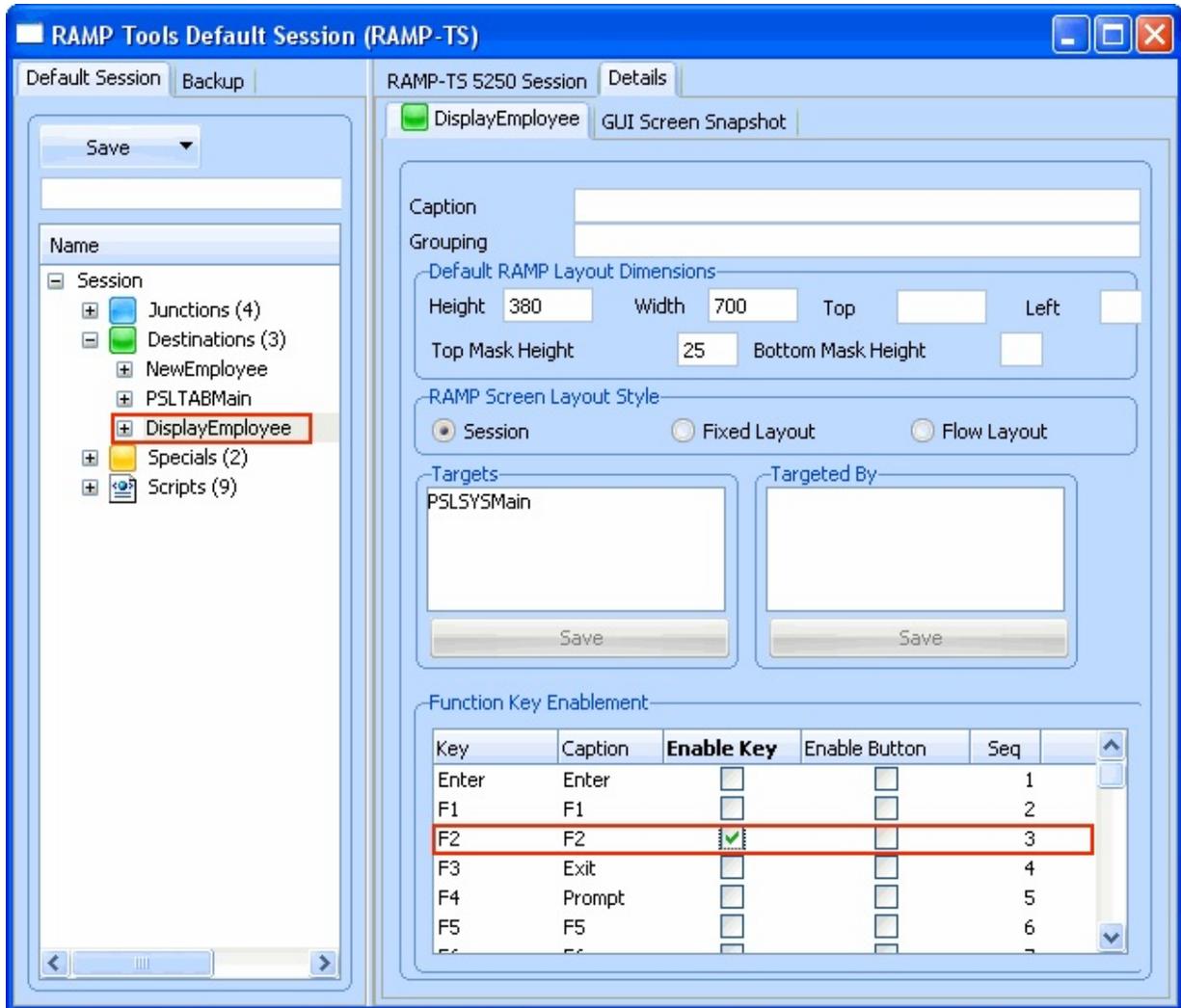
RAMP-TS010 Step 2. Associating the Field with the Handler

In this step you will associate the Department Code field with the special handler.

1. Select Session in the screens and scripts list. The session object is where you associate fields and special field handlers for all the screens in your session.
2. In the Special Field Handling table, specify the utxtDepartment field and a special handler named DF_PRM04. Use F2 as the key to invoke the handler.



3. Select the DisplayEmployee screen in the screens and scripts list.
4. Enable the F2 key.

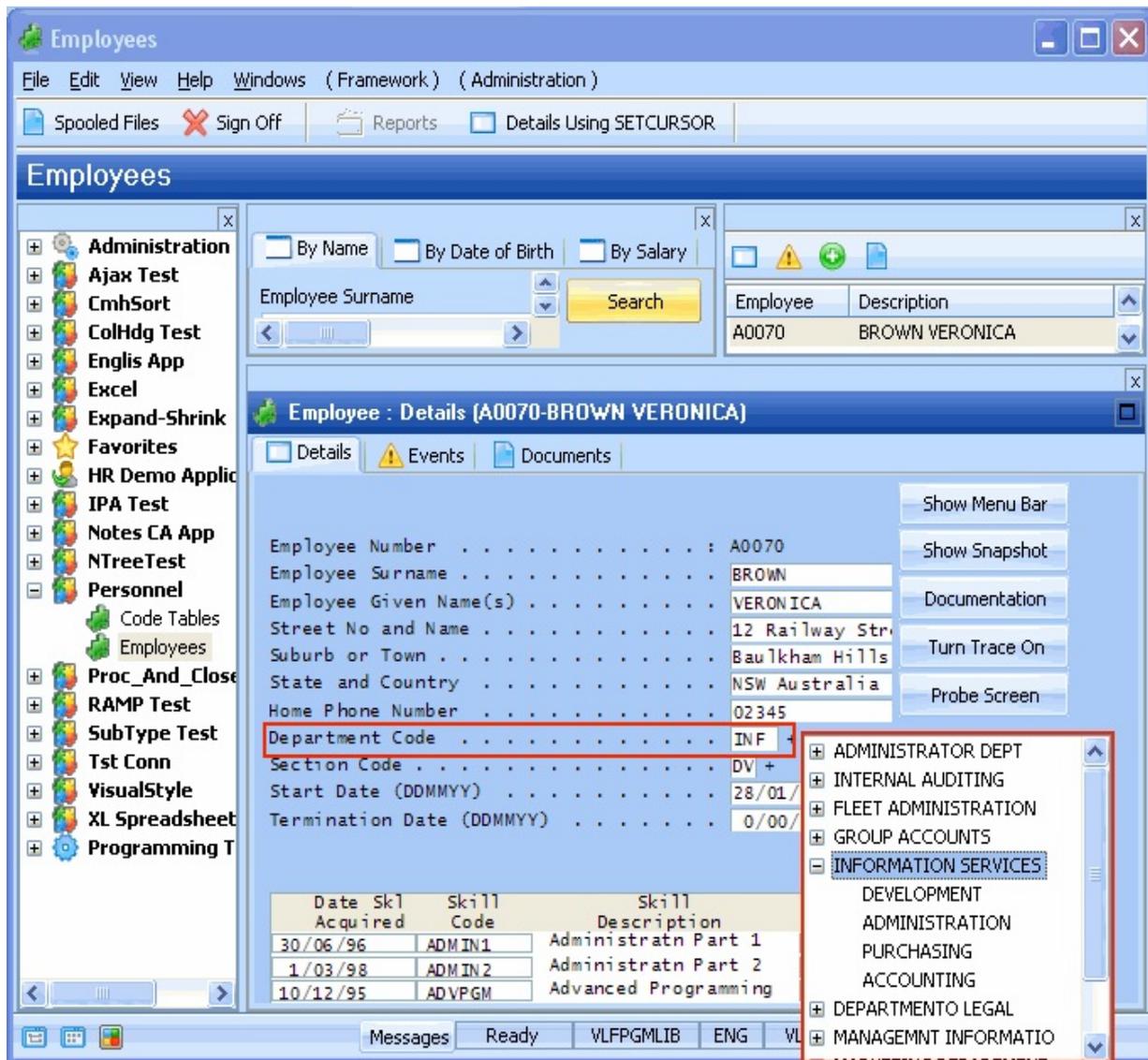


5. Do a partial save of the RAMP definition.

RAMP-TS010 Step 3. Test the Special Field Handler

In this step you will test the special field handler.

1. Switch to the Framework.
2. In the Personnel application, display the details of an employee.
3. Put the cursor on the Department Code field and press F2 to display the special field handler:



4. Double-click on a tree entry. The selected department code is inserted back into the field on your 5250 screen instantly. Because you have a field named

SECTION on your 5250 screen, it is also updated. This is because the sample field handler DF_PRM04 has code to handle a field named SECTION (you might want to have a look at the source code of this form).

5. Select items in the tree, without double-clicking. Notice that they are immediately updated back into your 5250 form.
6. Click back somewhere on your 5250 form to make DF_PRM04 go away.
7. Type "M" when the field handler is displayed. Notice the handler tries to guess the closest matching department.

Note: DF_PRM04 is a classic F4 prompter. How it behaves it is entirely up to you. By using it (and the other DF_PRMnn shipped examples) you can try out and modify various types of prompting so that you will know how to create your own special field handling components.

Summary

Important Observations

Special field handling is an advanced prompting facility for fields.

You specify the name of the field to be prompted, the function key to be used and the Visual LANSA form that is used as the prompter.

Unlike System i prompting, Visual LANSA prompter forms do not necessarily cause any interaction with the System i server which makes them fast.

Special field handling can be used to provide functionality that is not possible on a 5250 device.

For a more detailed tutorial about special field handling refer to [RAMP-TSAD03: Special Field Handling](#) .

Also see the topic [Advanced Prompting](#).

What You Should Know

How to associate special field handling with fields on modernized 5250 screens.

-

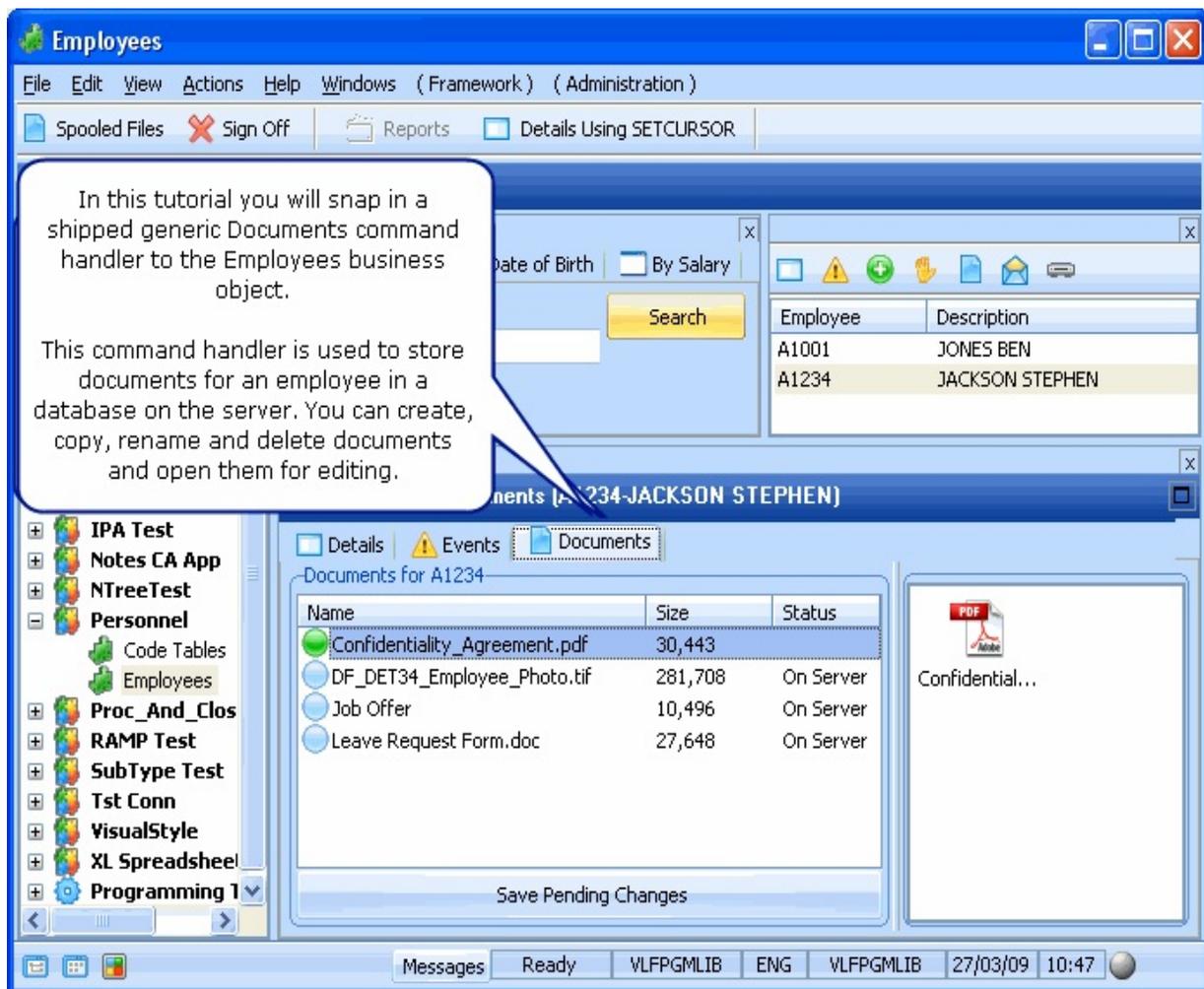
RAMP-TS011: Snapping in Shipped Documents Command Handler

In this tutorial you will learn how to snap in a shipped generic Documents command handler to your RAMP application.

Objectives

Learn to use a generic shipped command handler to your application

Understand how easily you can add value to a modernized application



To achieve this objective, you will complete the following steps:

[RAMP-TS011 Step 1. Snapping in the DX_DOCS Command Handler](#)

[RAMP-TS011 Step 2. Adding Documents](#)

[RAMP-TS011 Step 3. Working with Documents](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

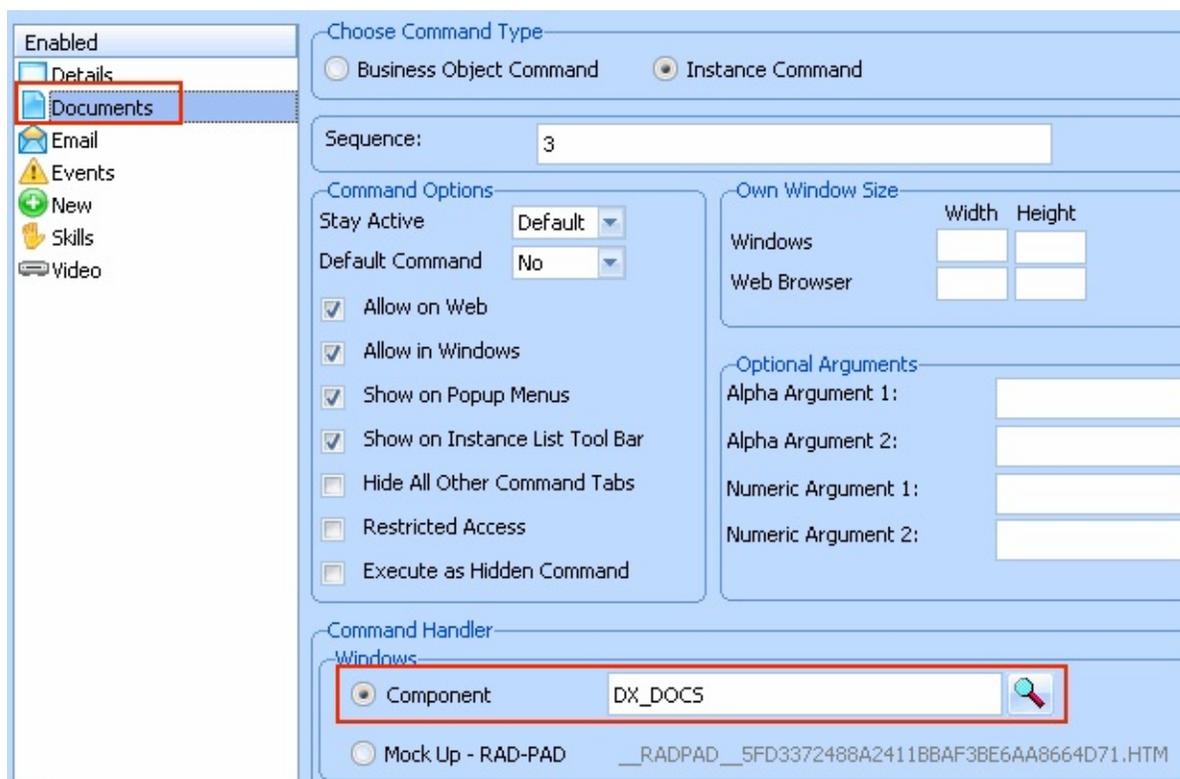
[RAMP-TS009: Tracing Navigation](#)

[RAMP-TS010: Using Special Field Handlers](#)

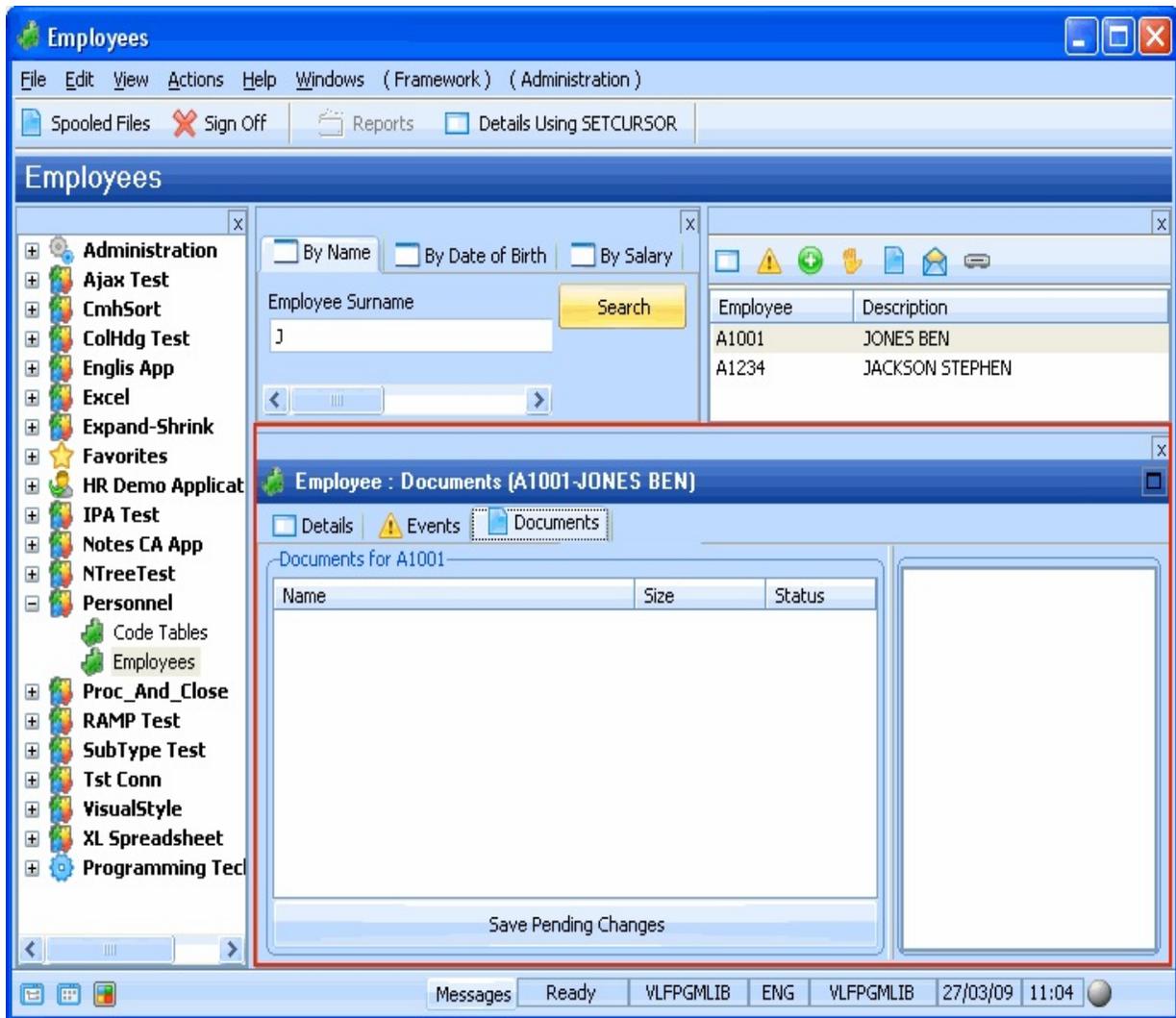
RAMP-TS011 Step 1. Snapping in the DX_DOCS Command Handler

In this step you will snap the shipped Documents command handler DX_DOCS to the Employees business object.

1. In the Framework window double-click the Employees business object to display its properties.
2. Display the Commands Enabled tab.
3. Specify DX_DOCS as the command handler for the Documents command:



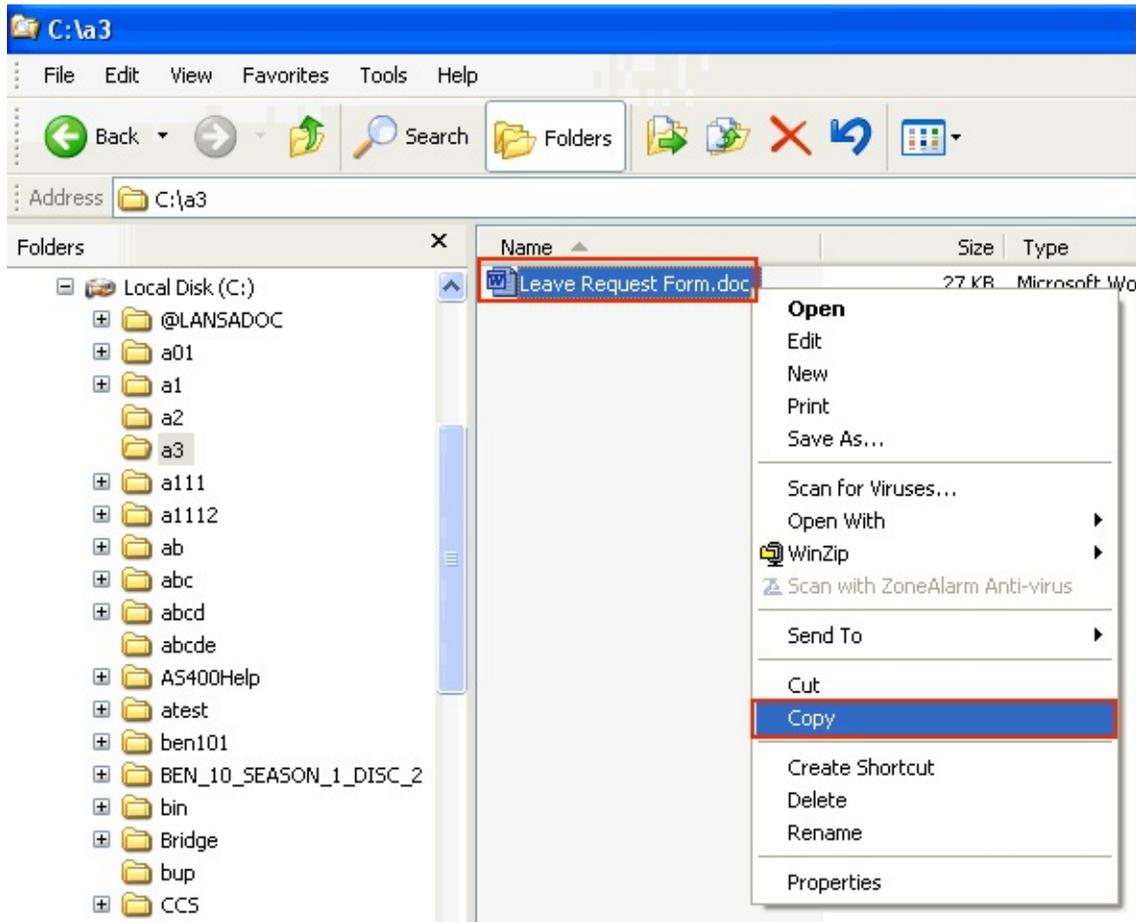
4. Close the properties of the Employees business object.
5. Select an employee in the instance list and display the Documents tab. The shipped documents command handler is snapped in and usable:



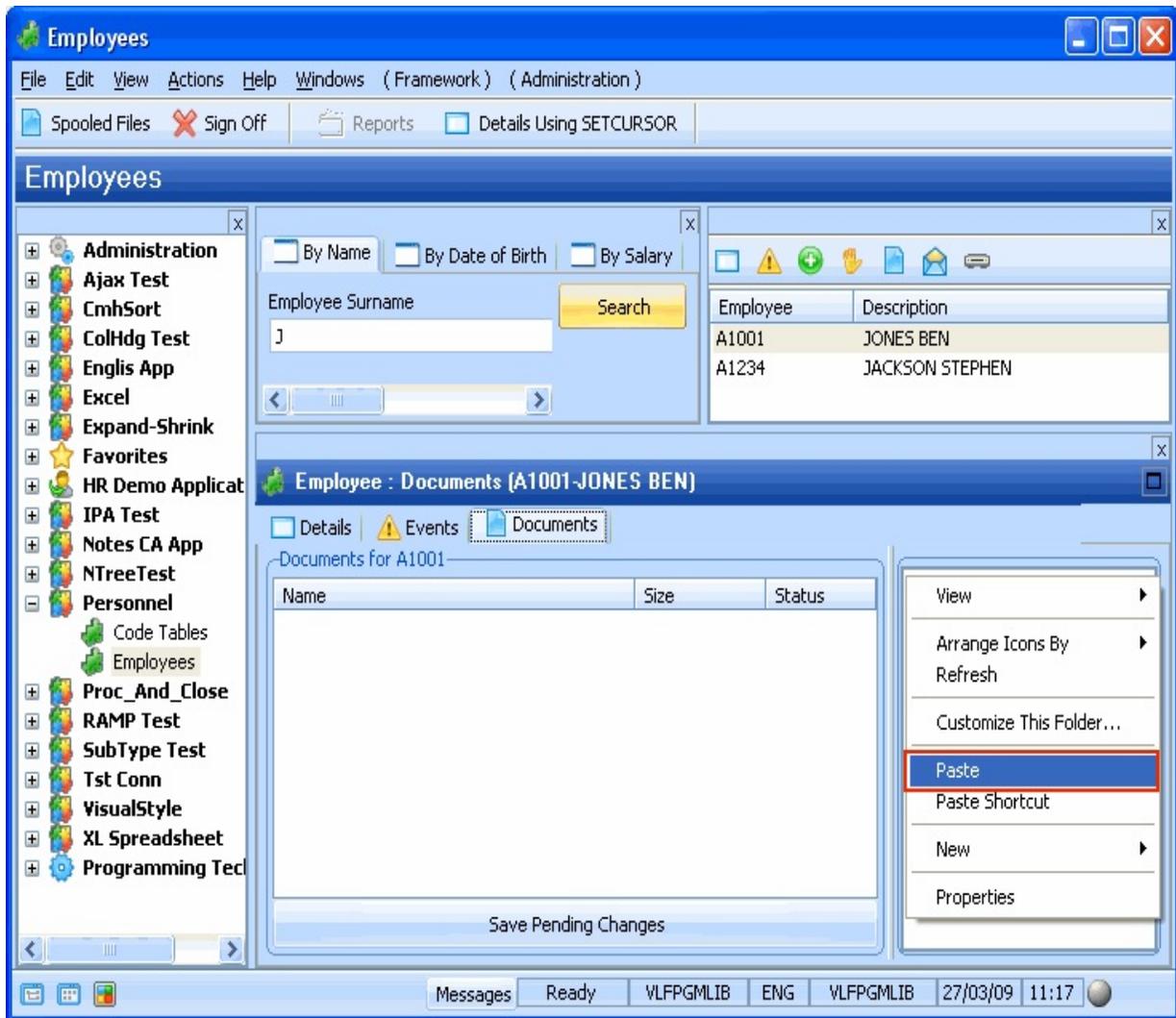
RAMP-TS011 Step 2. Adding Documents

In this step you will learn how to use the shipped Documents command handler.

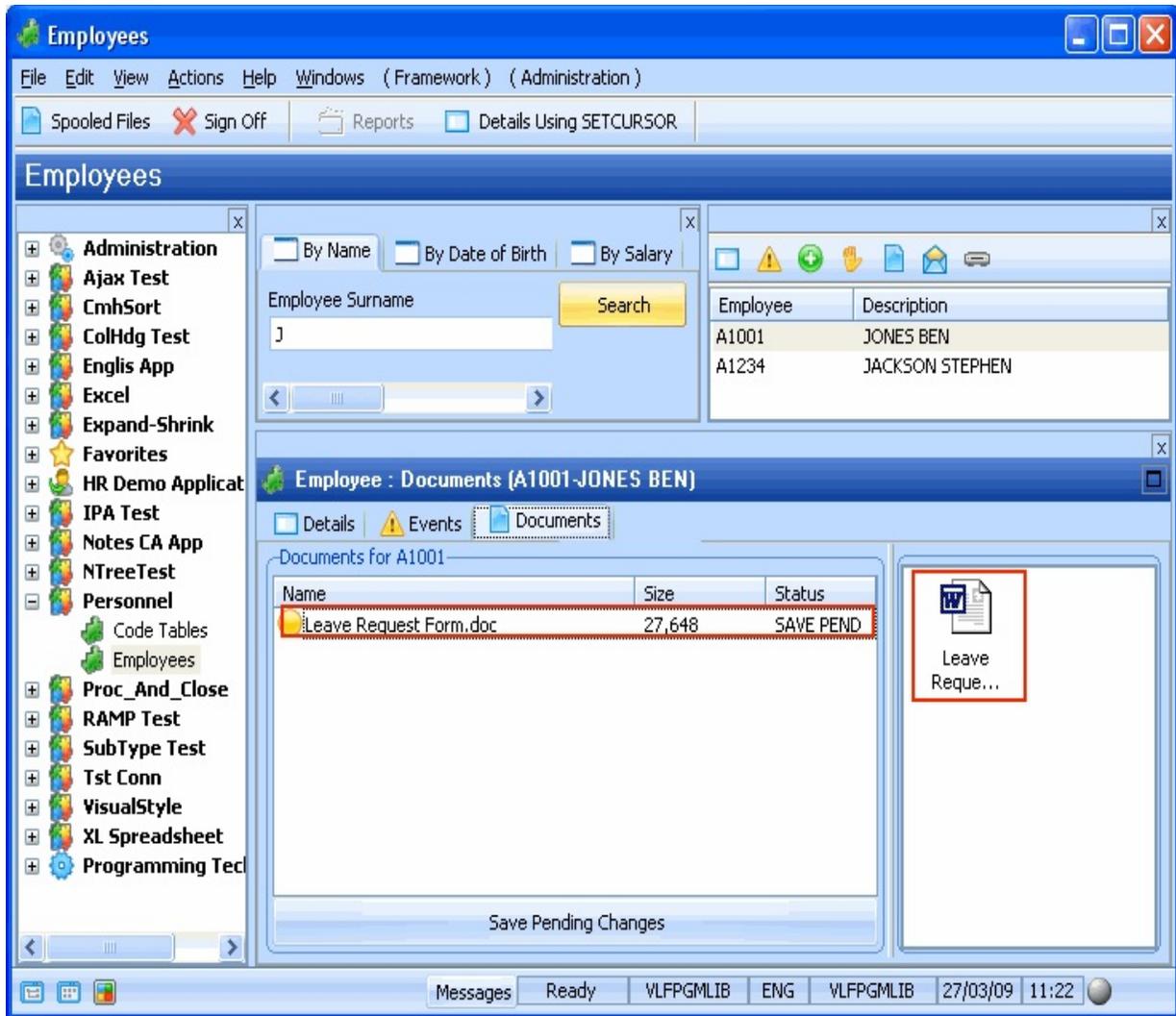
1. Copy a document (for example a Word document or a PDF) in Windows Explorer:



2. Display the Documents tab in your Framework.
3. Right-click the area on the right of the Documents command handler and select Paste from the context menu:



The document is added to the command handler:



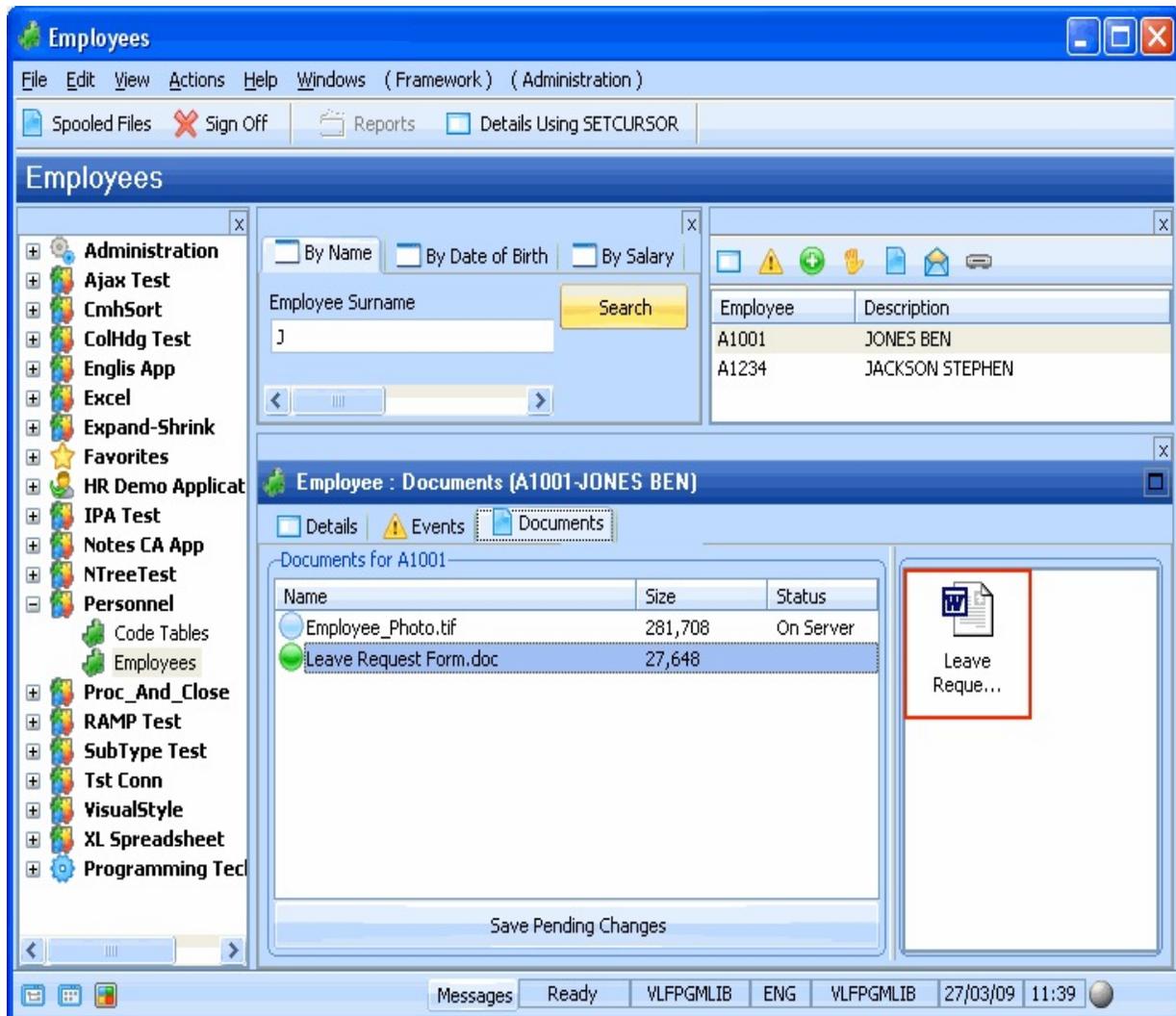
4. Click on the Save Pending Changes button to store the document in the shipped database file DXDOCS on the server.
5. Add another file, for example a photo or another image to the command handler and save it.

Note that you can also use drag-and-drop to add documents, or use the context menu to create new documents..

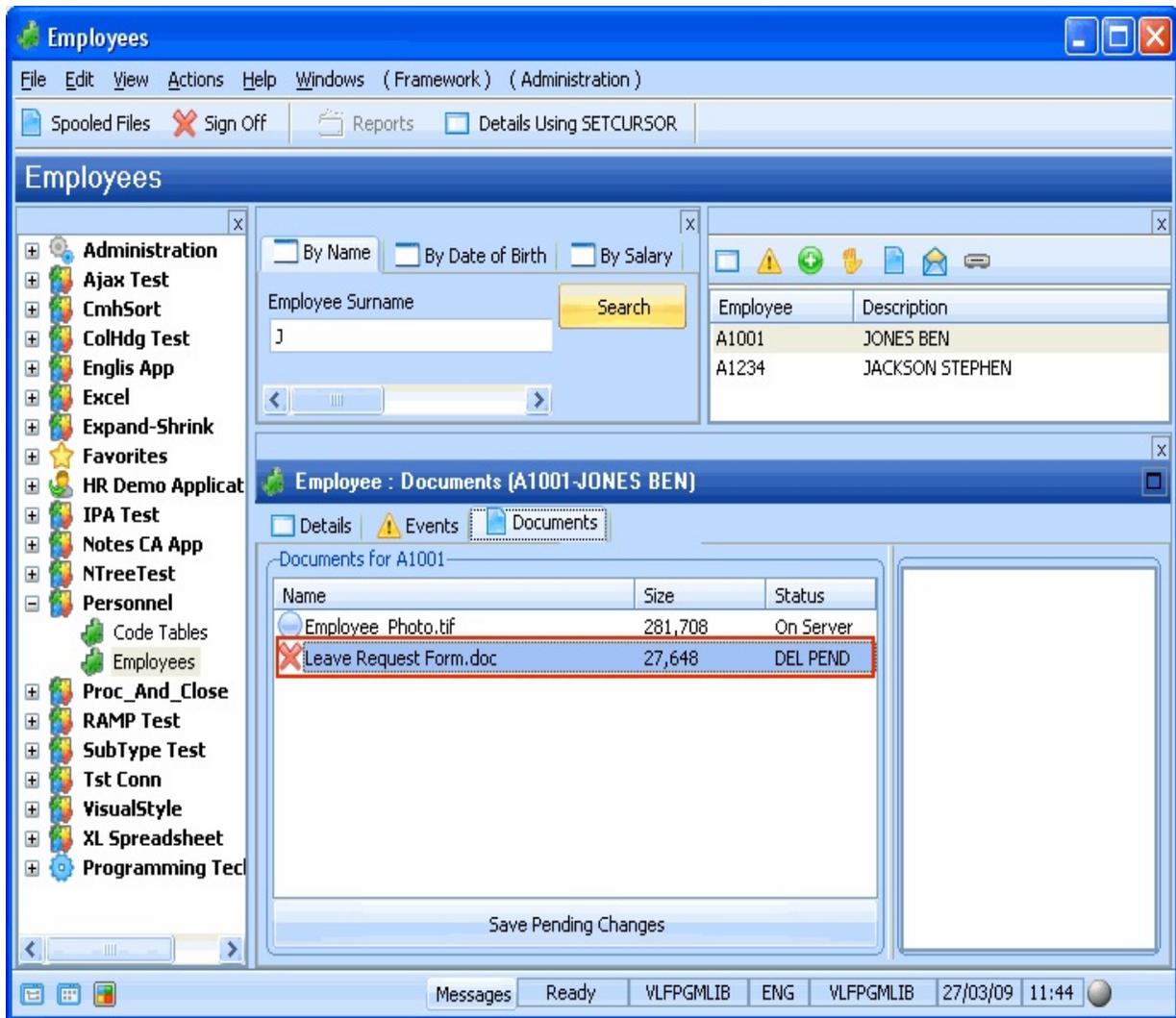
RAMP-TS011 Step 3. Working with Documents

In this step you will learn how to edit and delete documents in the Documents Command handler.

1. Select a document from the list of documents. Notice that it is displayed on the area on the right where you first dropped it.



2. Double-click the document to open it.
3. Close the document and display the Framework if it is not showing.
4. Select the document and right-click to display the context menu.
5. Choose Delete. Notice that Documents command handler indicates that the delete is pending.



6. Click on the Save Pending Changes button to delete the document.

Summary

Important Observations

Reusing shipped command handlers may add significant value to any 5250 application that is being RAMPed.

The Documents command handler can be used with any business object.

In the Documents command handler you can create, delete, rename and copy documents and open them for editing.

You can use copy and paste or drag-and-drop to add documents.

An icon next to the list entry in the Documents command handler indicates the status of the document.

What You Should Know

How to use the shipped generic Documents command handler DX_DOCS

-

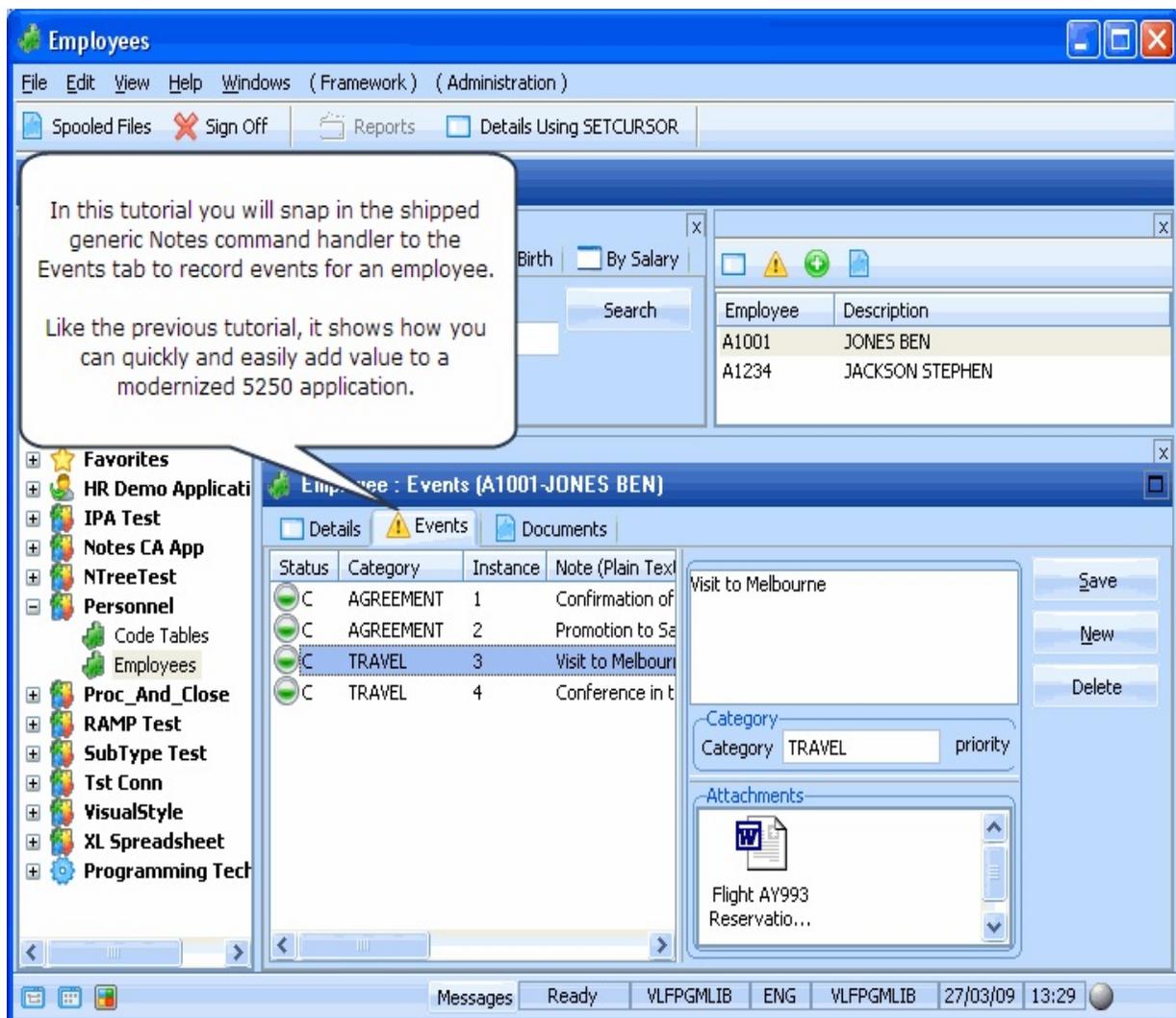
RAMP-TS012: Snapping in Shipped Notes Command Handler

In this tutorial you will learn how to use the shipped generic Notes command handler.

Objectives

To see another example of how easy it is to quickly add value to a modernized application

To start thinking of how to extend the use of generic command handlers to other business objects.



To achieve this objective, you will complete the following steps:

[RAMP-TS012 Step 1. Snapping in the DF_T3201 Command Handler](#)

[RAMP-TS012 Step 2. Adding Notes](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

[RAMP-TS009: Tracing Navigation](#)

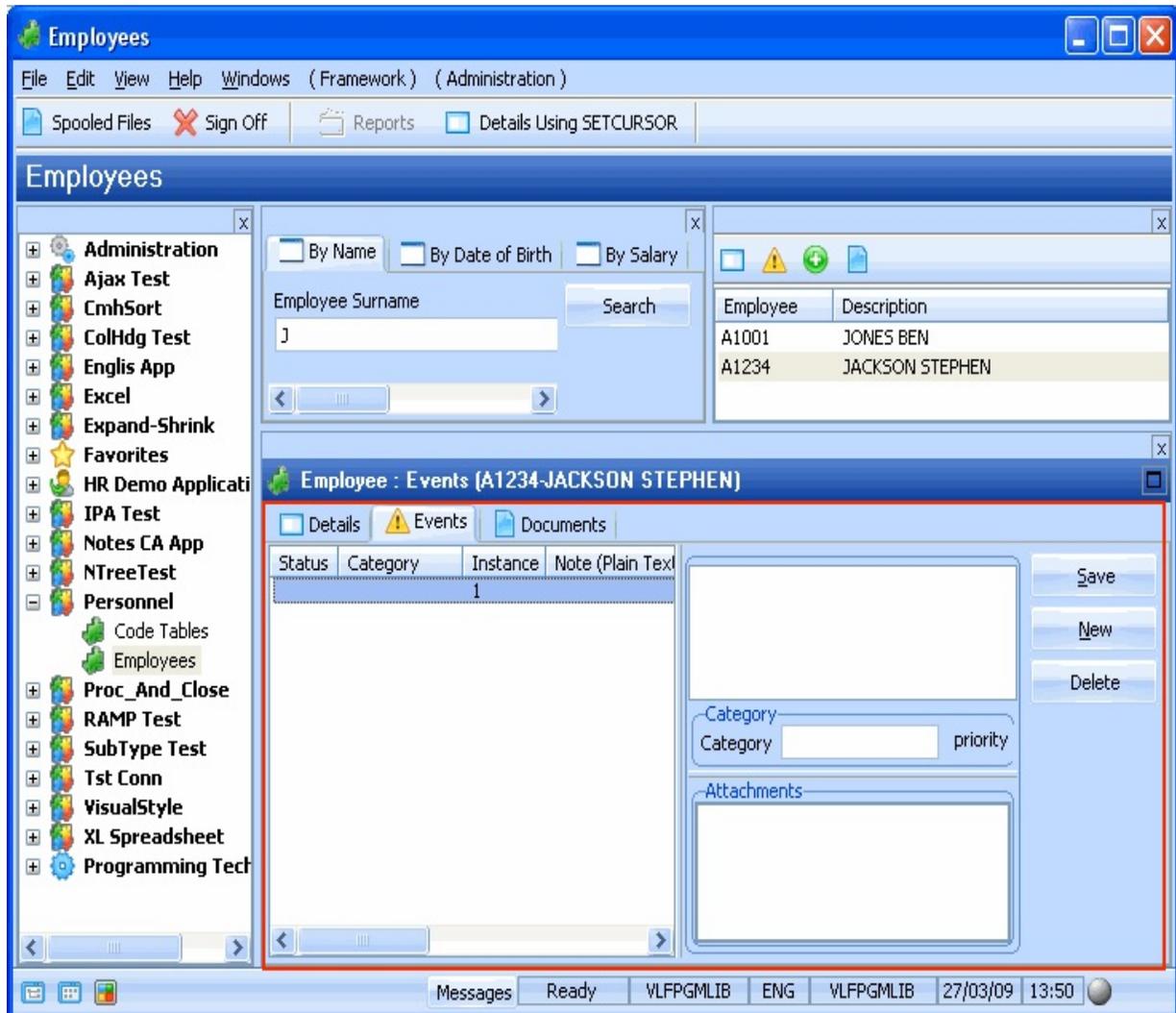
[RAMP-TS010: Using Special Field Handlers](#)

[RAMP-TS011: Snapping in Shipped Documents Command Handler](#)

RAMP-TS012 Step 1. Snapping in the DF_T3201 Command Handler

In this step you will associate the generic Notes command handler with the Events command of the Employees business object.

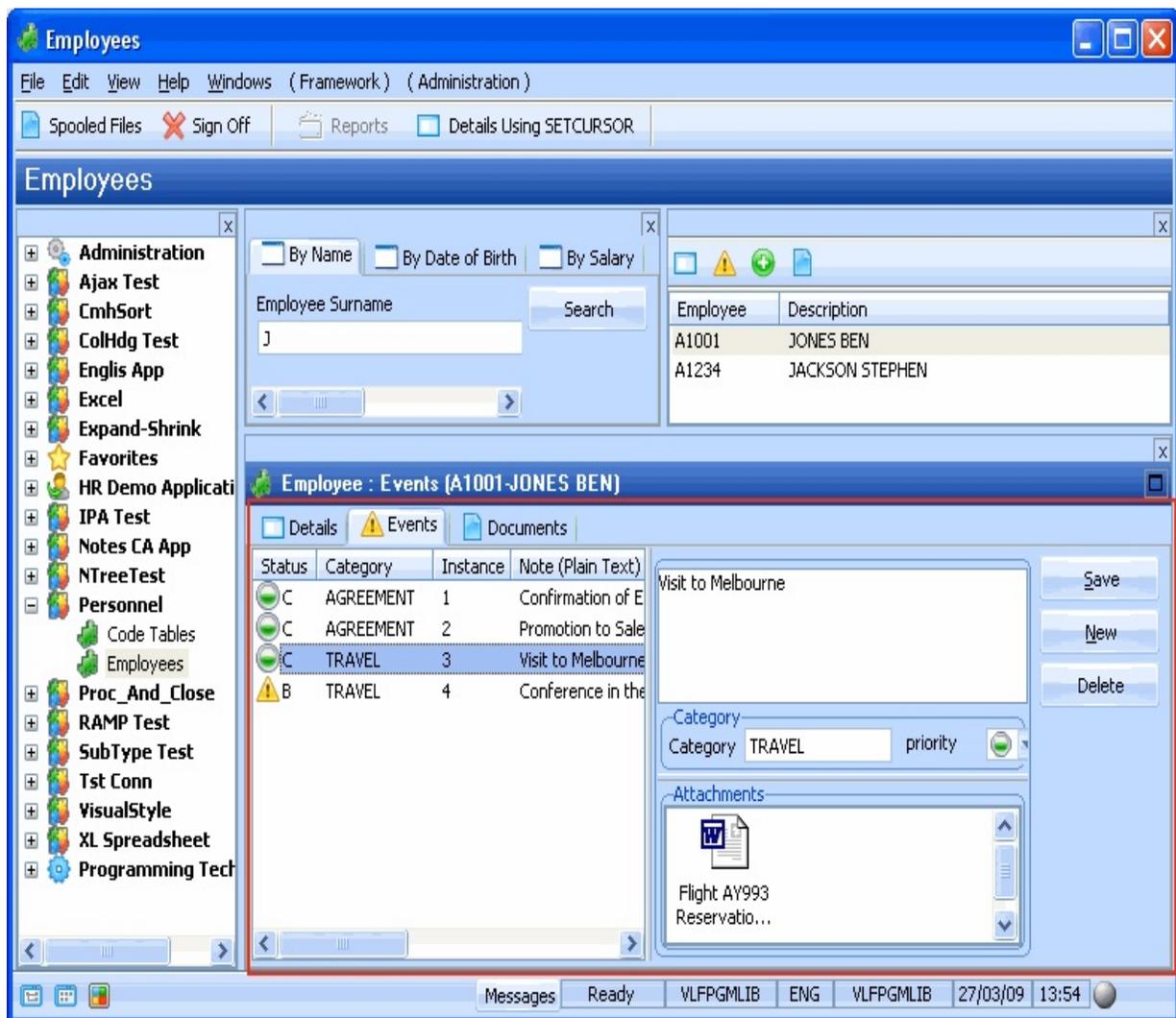
1. Display the properties of the Employees business object.
2. In the Commands Enabled tab, associate the Events command with command handler DF_T3201.
3. Close the properties of the Employees business object.
4. Select an employee from the instance list and display the Events tab. The shipped notes command handler is snapped in and usable:



RAMP-TS012 Step 2. Adding Notes

In this step you will learn how to add notes for an employee. You can optionally also attach documents to the notes, categorize them and set their priority.

1. Add a few notes for an employee using the Save and New buttons.
2. Add an attachment to a note by cut-and-paste or drag-and-drop as in the previous tutorial.
3. Create categories for the notes and set their priority. For example:



Summary

Important Observations

Reusing shipped command handlers may add significant value to any 5250 application that is being RAMPed.

The Notes command handler can be used with any business object.

The command handler makes it possible for the end-user to enter a number of notes against any instance of the business object

The note is saved as a string field on database file FPNOTE

The attachment documents are saved on database file FPDOC

Attachment documents can be added to a note

A category value can be specified for the note to allow the user to sort the notes according to their own criteria

User Created/Updated and Date/Time Created/Updated are automatically recorded on the list

What You Should Know

The ideas presented in this tutorial can easily be extended to other business objects.

For example a Product business object might have a press release, a brochure, a price list associated with it. It might also have many events associated with it, such as launch, customer complaints, recalls, end of life, etc., each of which may have many documents associated with it.

There is also a shipped generic Command Handler for spool files, DF_T3101.

RAMP-TS013: Sending Instance List Data to Excel

This tutorial will show how to integrate your application with Microsoft Excel.

Objectives

Learn how to create a command handler that sends data to Microsoft Excel.

To achieve this objective, you will complete the following steps:

[RAMP-TS013 Step 1. Creating the Command Handler](#)

[RAMP-TS013 Step 2. Snapping in and Testing the Command Handler](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

[RAMP-TS009: Tracing Navigation](#)

[RAMP-TS010: Using Special Field Handlers](#)

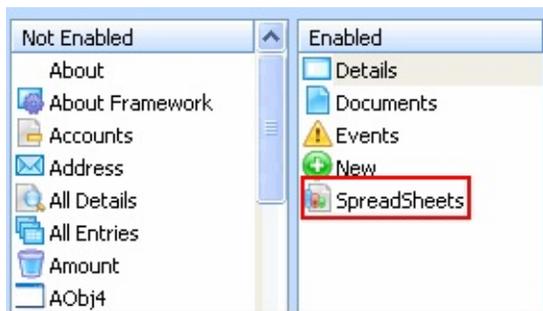
[RAMP-TS011: Snapping in Shipped Documents Command Handler](#)

[RAMP-TS012: Snapping in Shipped Notes Command Handler](#)

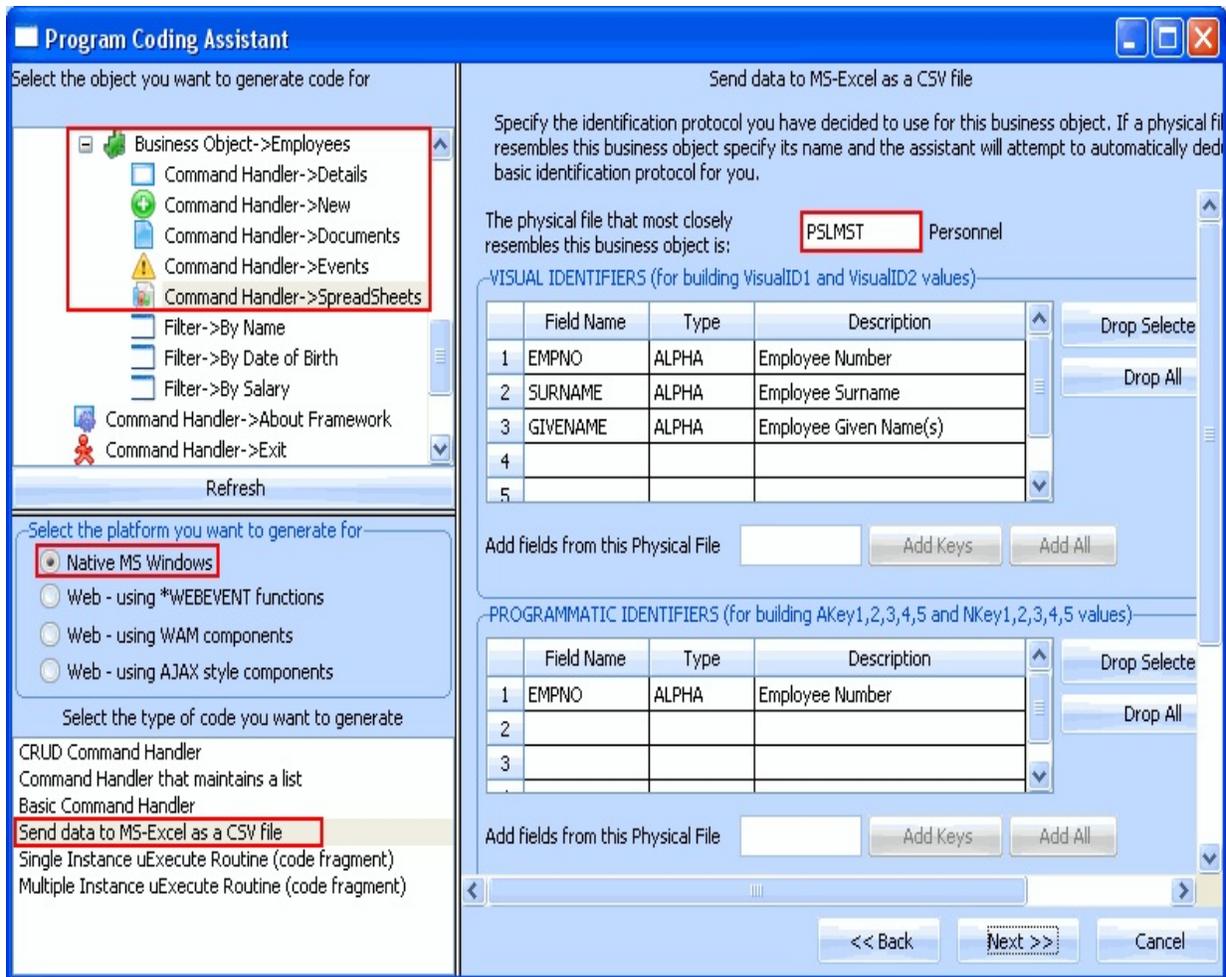
RAMP-TS013 Step 1. Creating the Command Handler

In this step you will use the Program Coding Assistant to create the Spreadsheets command handler that can be used to send data to Microsoft Excel.

1. In the Framework window, display the properties of the Employees Business object.
2. Display the Commands Enabled tab.
3. Enable the Spreadsheets command.

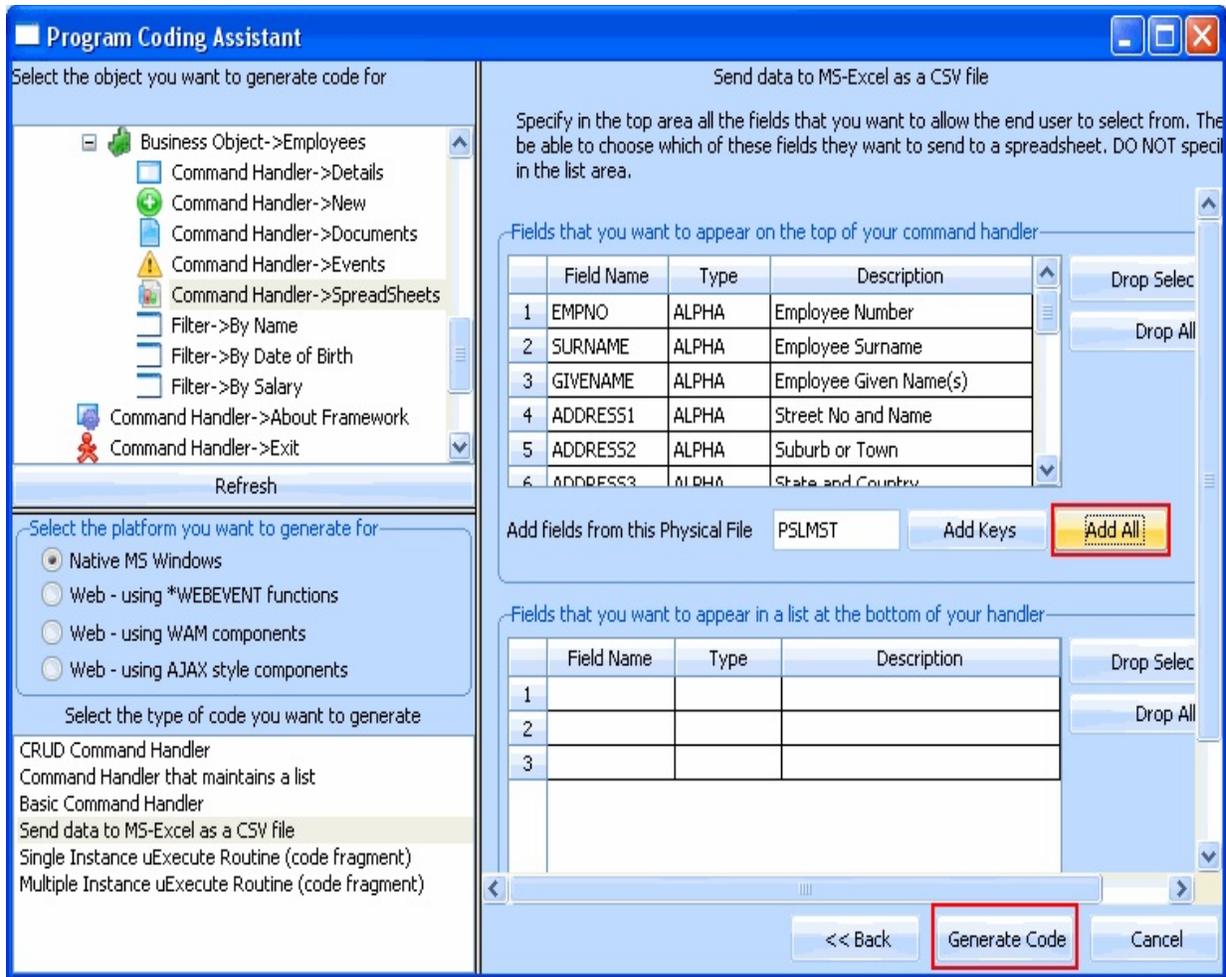


4. Close the properties of Employees.
5. Start the Program Coding Assistant from the Framework menu.
6. Select the Employees business object and the Spreadsheets command handler.
7. Select Native MS Windows as the platform.
8. Select Send data to MS-Excel as a CSV file as the type of code you want to generate.
9. Click Next.
10. Select PSLMST as the physical file and accept the default visual and programmatic identifiers.



11. Click Next.

12. Select to include all the fields from the PSLMST file on the top of the command handler.



13. Click Generate Code.

The next page, Generated Code, displays the source code for your command handler. You now need to create the component that will contain this code:

14. Specify iiiRMP02 as the name of your command handler and Spreadsheet Command Handler as the description. (iii are your initials).

15. Click on the Create button to create the component.

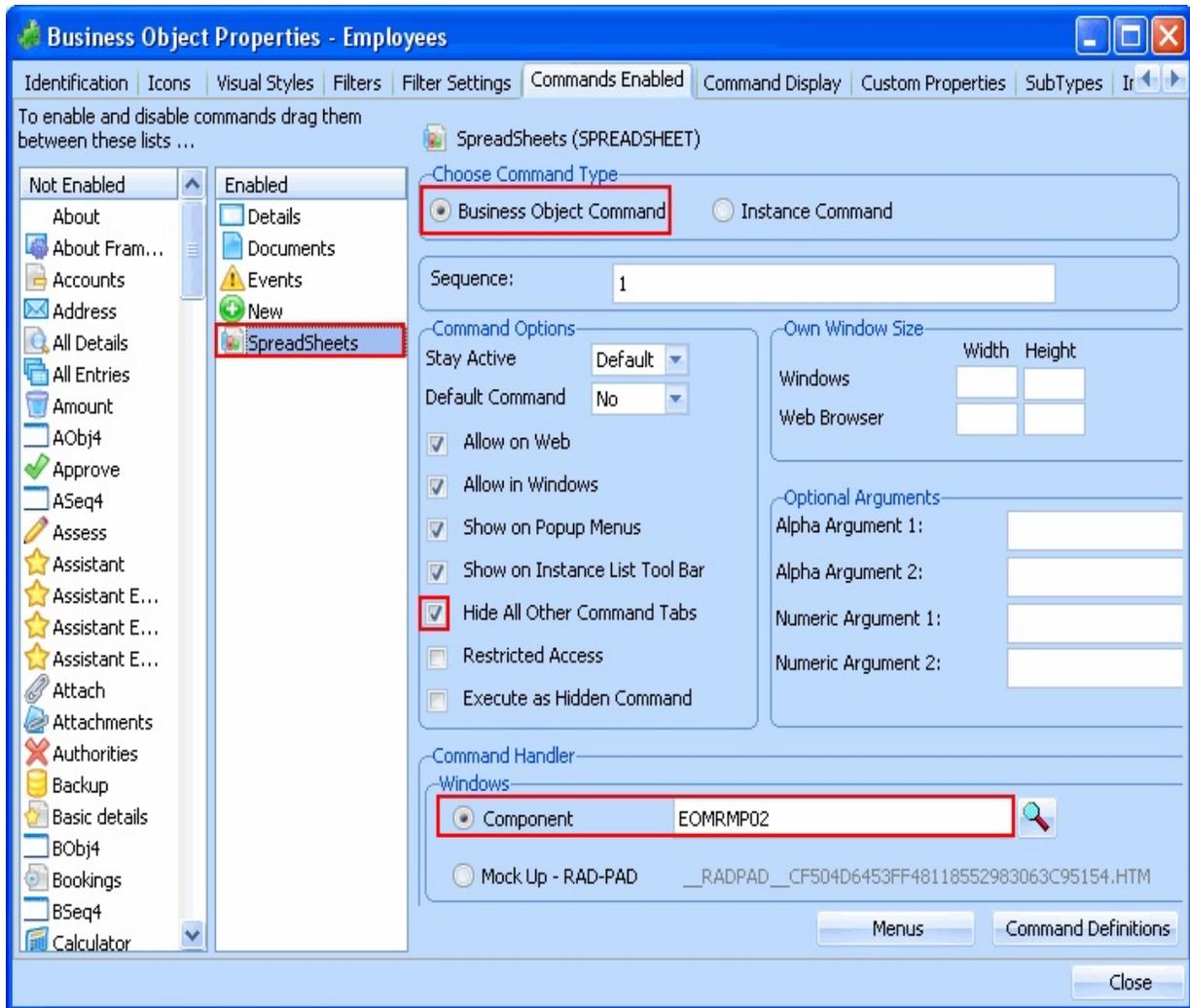
After a brief delay the command handler component is displayed in the Visual LANSAs editor.

16. Compile the component in the Visual LANSAs editor.

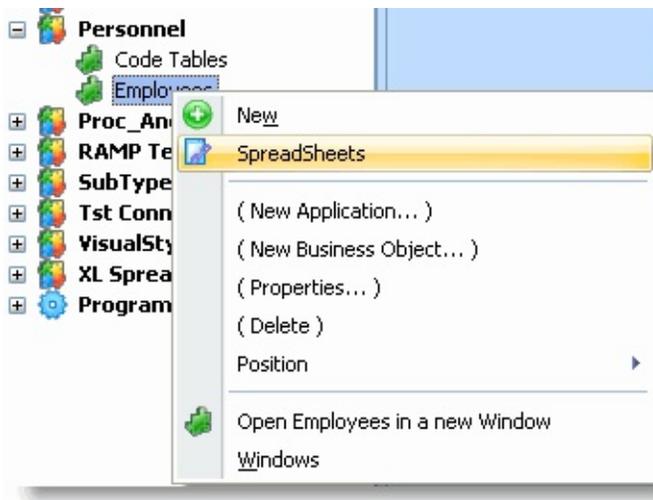
RAMP-TS013 Step 2. Snapping in and Testing the Command Handler

In this step you will snap in the Spreadsheets command handler to the Framework and test it.

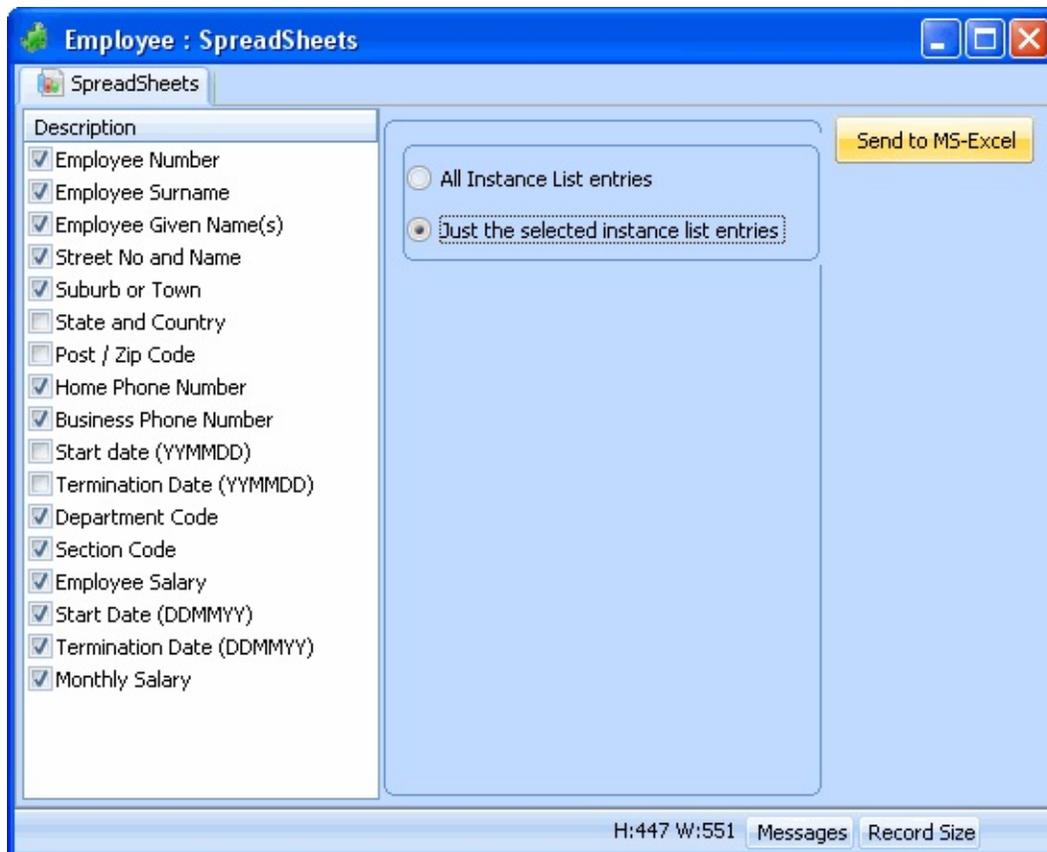
1. In the Framework window, display the properties of the Employees business object.
2. Display the Commands Enabled tab.
3. Select the Spreadsheets command handler.
4. Select the option Business Object Command (the command handler can be used for all employees, not just one employee instance).
5. Select the Hide All Other Command Tabs option to ensure that the command tab for New is not displayed in the window.
6. Specify the name of your command handler (iiiRMP02) as the Windows component.



7. Close the properties of the employees business object.
8. Save and restart the Framework.
9. Use the filter of the Employees business object to select employees.
10. Select some employees in the instance list (hold the Ctrl key down and click with the mouse).
11. Then right-click Employees in the navigation pane or right-click an employee in the instance list and select Spreadsheets from the context menu.



12. In the command handler, select the fields to be sent to Excel and the option Just the selected instance list entries.



13. Click the Send to MS Excel button.

An MS Excel spreadsheet with the selected employee data is displayed:

Employee Number	Employee Name	Employee Name	Street No	Suburb or	Home Pho	Business P	Departmer	Section Co	Employee	Start D
A1031	BLOGGS	JOHN	3 Woodbur	Winston H	(02) 9668 9	(02) 9922 4	MIS	EI	6072500	180
A2006	BRYERS	MIKE	14 Napier	North Sydn	98888888	9289889	ADM	2	50000	90
A0070	BROWN	VERONIC	12 Railway	Baulkham	TRN	(02) 9647 2	INF	DV	5012500	280

You can now work with the employee data in Excel.

Summary

Important Observations

Integrating the application with desktop tools such as Excel adds real business value to the 5250 application because it provides new information and capabilities to the application users - unlike, for example, a drop down, which adds very little real business value to a 5250 application.

This feature allows you to easily leverage the power of MS-Excel. Once a user has information in MS-Excel they can do what they please with it – draw graphs and charts, produce pivot tables, save it to their hard drive, print it, send it via e-mail to others. MS-Excel provides a gateway for using valuable information locked up in your IBM i data base. MS-Excel is also a great springboard for generic reporting activities.

The spreadsheet data extraction is implemented by software developers - rather than by end users with ad-hoc tools - so more it is more secure and its performance has been verified.

What You Should Know

How to create a command handler that sends data for all or selected instance list entries to MS Excel.

RAMP-TS014: Snapping RAMP Screens into the HR Demo Application

In this tutorial you will add your newly created RAMP screens to the HR Demo Application.

Objectives

Learn how to integrate RAMP screens and Framework components in an application.

Understand more about how the instance list and RAMP command handlers interact.

In this tutorial you will snap in your RAMP screens to the HR Demo Application in order to see how easily you can mix and match modernized 5250 screens with Framework components.

The screenshot shows the 'Resources' application window. The main area displays a table of employee records. The selected record is for employee A1001, BEN JONES, residing at 144 Frog Pybble, NSW.

Code /...	Address...	Address...	Address...	Zip Code	Business...	Home...	D...	S...	
CA A0070	12 Ra...	Bauk...	NSW ...	2153	(02) ...	TRN	I...	DV	
H... A0090	70 M...	NEW...	AUST...	2220	654 6...	344-2...	F..	03	
A1001	144 F...	PYMB...	NSW,	2001	798 0...	799 5...	A..	01	
A1003	29 Ar...	DEE ...	NSW,	2000	406 6...	977 6...	F..	02	
... A1015	59 Da...	BEXLEY,	NSW,	2030	789 4...	450 1...	A..	01	
ROBINSON, MARY	A1025	14 W...	ST IV...	NSW,	2005	456 1...	126 3...	A..	01

The detailed view for employee A1001 shows the following information:

- Employee Number: A1001
- Employee Surname: JONES
- Employee Given Name(s): BEN
- Street No and Name: 144 Frog
- Suburb or Town: PYMBLE.
- State and Country: NSW.
- Home Phone Number: 799 5268
- Department Code: ADM +
- Section Code: 01 +

Buttons on the right side of the detailed view include: Save, Prompt, Probe Screen, Show Snapshot, Documentation, and Turn Trace On.

To achieve this objective, you will complete the following steps:

[RAMP-TS014 Step 1. Snap in RAMP Screens to the HR Demo Application](#)
[RAMP-TS014 Step 2. Modifying the SETVALUE Statement](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

[RAMP-TS009: Tracing Navigation](#)

[RAMP-TS010: Using Special Field Handlers](#)

[RAMP-TS011: Snapping in Shipped Documents Command Handler](#)

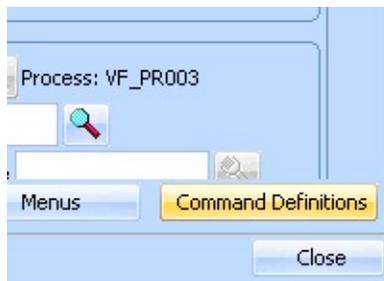
[RAMP-TS012: Snapping in Shipped Notes Command Handler](#)

[RAMP-TS013: Sending Instance List Data to Excel](#)

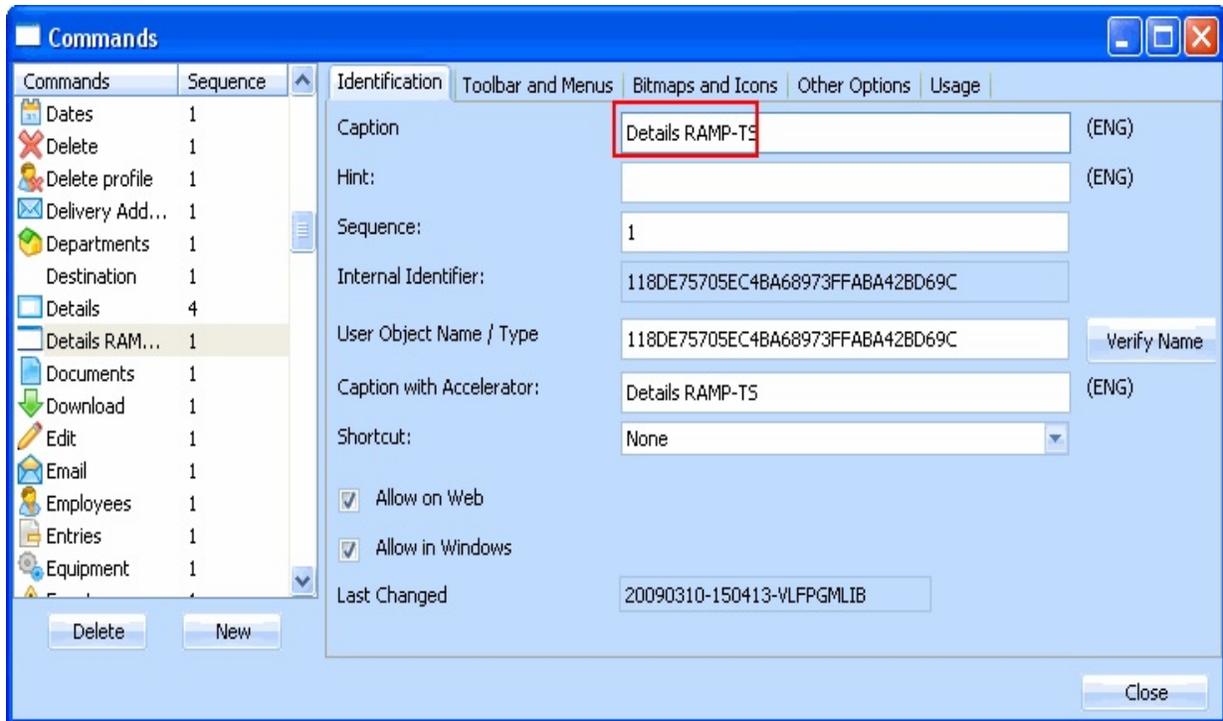
RAMP-TS014 Step 1. Snap in RAMP Screens to the HR Demo Application

In this step you will snap in the New Employee and DisplayEmployee screens to the HR Demo Application.

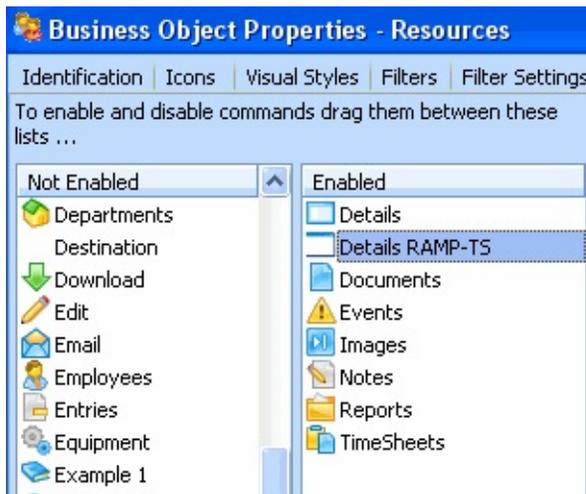
1. In the Framework, select the HR Demo Application and then the Resources business object.
2. Display the properties of the Resources business object.
3. Display the Commands Enabled tab.
4. Click on the Command Definitions button to add a new command:



5. In the Commands window click New to create a new command.
6. Make the command caption Details RAMP-TS.

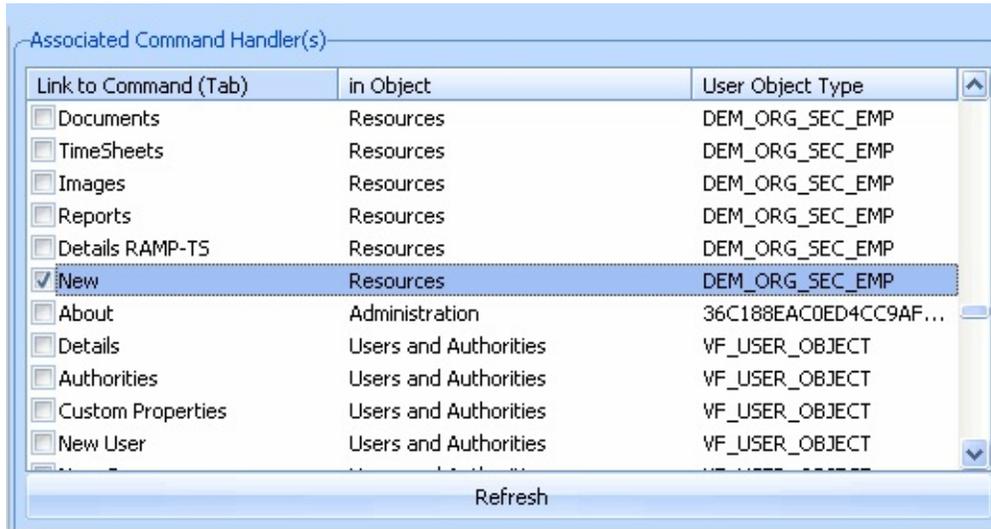


7. Close the Command Window.
8. Back in the Commands Enabled tab Drag the Details RAMP-TS command to the Enabled list.

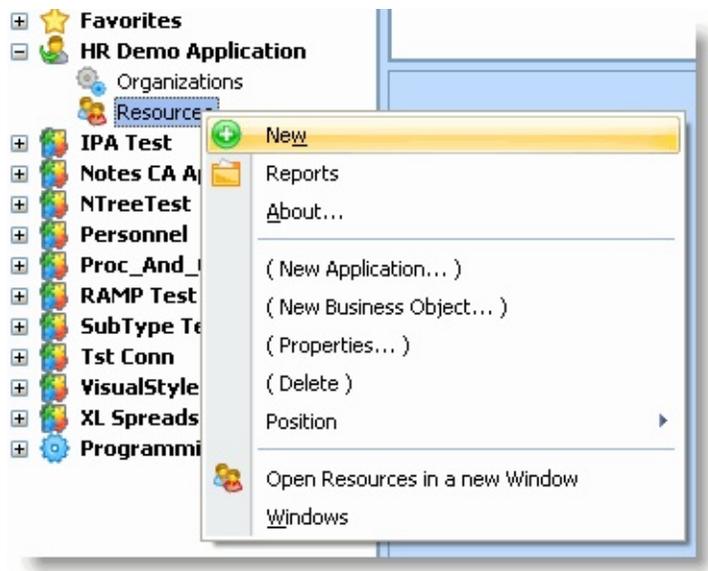


9. Drag the New command to the Enabled list.
10. Save and restart the Framework.

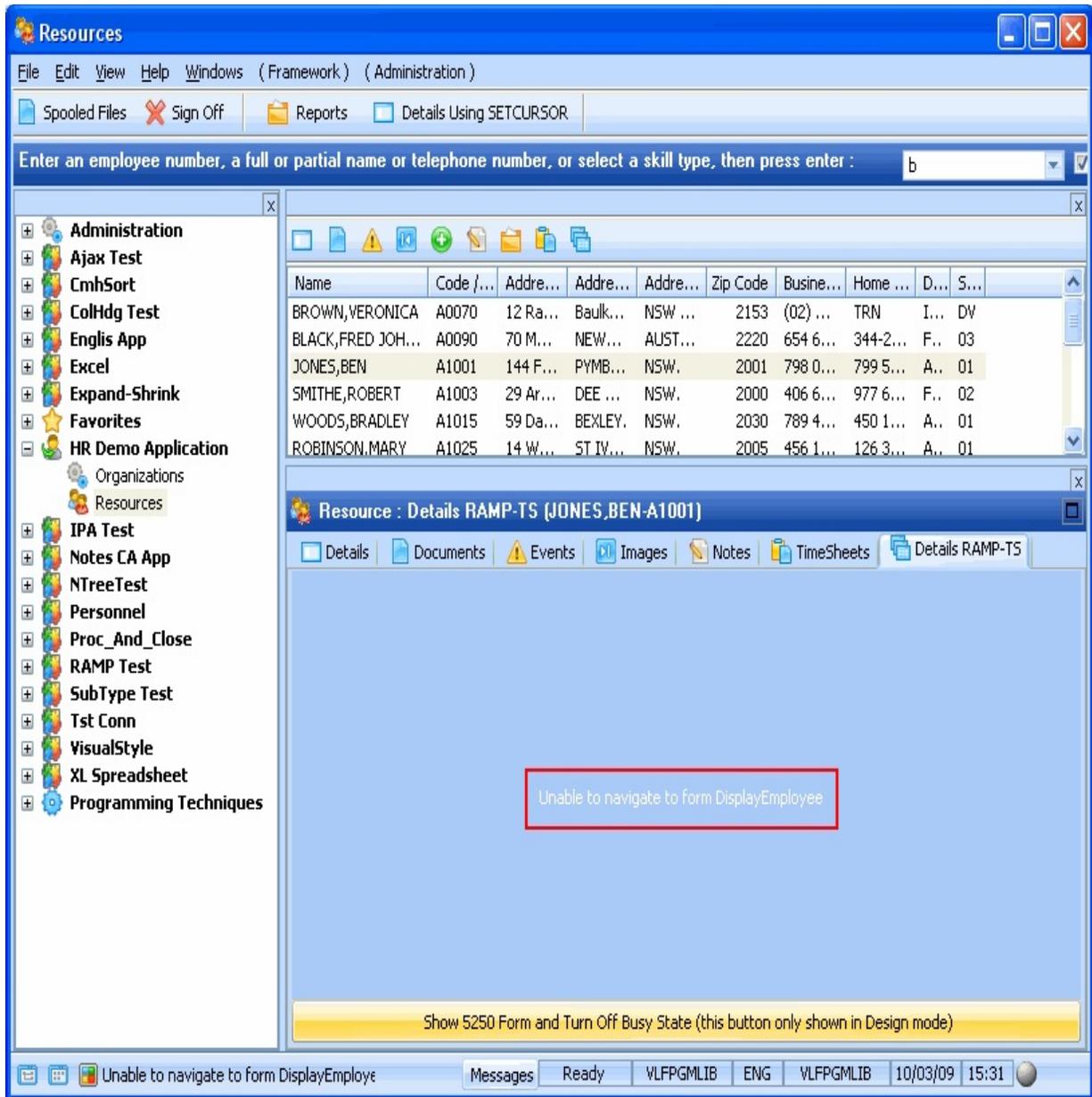
11. Start RAMP Tools. You do not need to start a 5250 identification session.
12. In the RAMP Tools window select NewEmployee in the screens and script list.
13. In the Associated Command Handlers associate the screen with the New command for the Resources business object.



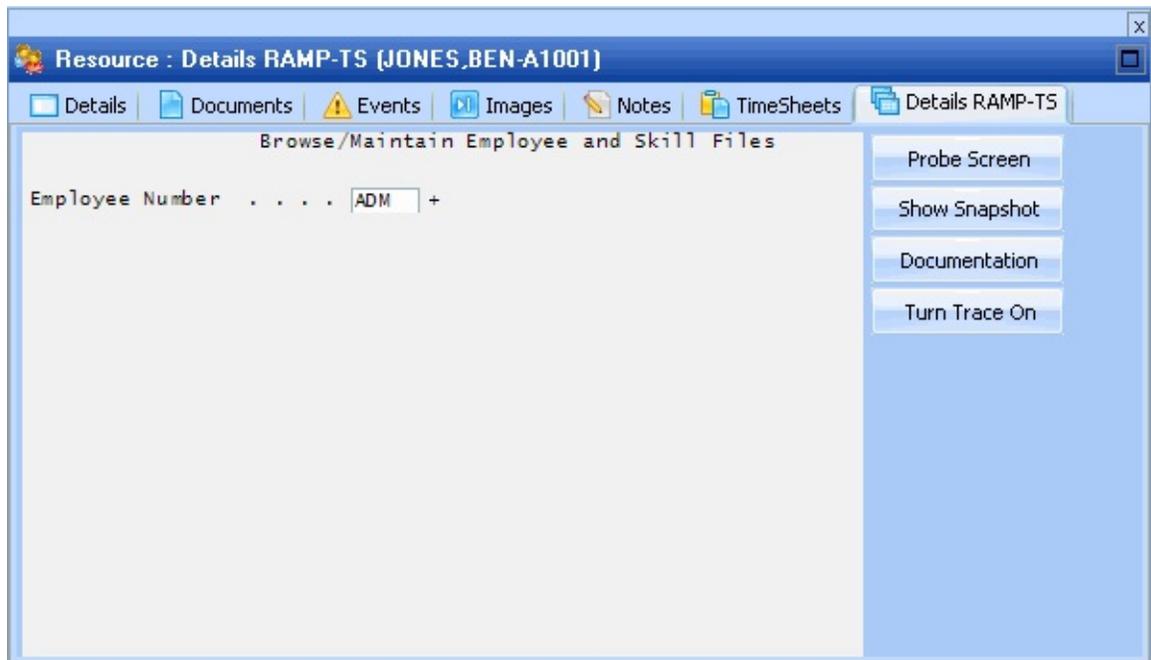
14. Then associate the DisplayEmployee screen with the Details RAMP-TS command of the Resources business object.
15. Do a partial save.
16. In the Framework window select HR Demo Application and the Resources business object.
17. Right-click and select New from the context menu to display the NewEmployee screen:



18. Close the NewEmployee screen.
19. Use the filter to display employees in the instance list.
20. Select an employee and click on the Details RAMP-TS command handler.
You will get an error Unable to navigate to DisplayEmployee.



21. Click on the Show 5250 Form and Turn Off Busy State button to display the screen where the navigation has stopped.



It is the FindEmployee screen. Notice that the Employee Number field has a department code as its value. In the next step you will change your script to retrieve the employee number from the instance list.

RAMP-TS014 Step 2. Modifying the SETVALUE Statement

In this step you will examine the filter of the HR Demo Application and modify the script associated with the FindEmployee screen so that it can be used in multiple locations in the Framework.

1. Display the properties of the Resources business object.
2. Select the Filters tab and then the Filter Snap-in Settings tab. Notice that the filter is DF_FILT9. The filter determines the Akey values used to fill the instance list.



3. Close the properties of Resources.
4. Close the Framework.
5. Switch to the Visual LANSA editor and locate and open reusable part DF_FILT9.
6. Search for the AddtoList method in the filter source:

```
Invoke Method(#avListManager AddtoList) BusinessObjectType(DEM_ORG_SEC_EMP) Visualid1(#FULLNAME)
Visualid2(#EmpNo) Akey1(#deptment) Akey2(#section) Akey3(#Empno) NColumn1(#PostCode)
AColumn1(#ADDRESS1) AColumn2(#Address2) AColumn3(#Address3) AColumn4(#PhoneBus)
AColumn5(#PhoneHme) AColumn6(#Deptment) AColumn7(#Section) NColumn2(#AdditionTotal)
SetasCurrent(#AdditionOption) ExecuteDefaultCmd(#AdditionOption)
```

7. Examine the Akey values in the method. Notice that the Empno field is Akey3.

Remember that the script associated with the FindEmployee screen uses the Akey1 value to get the employee number from the instance list (because it is the Akey value used in the By Name filter).

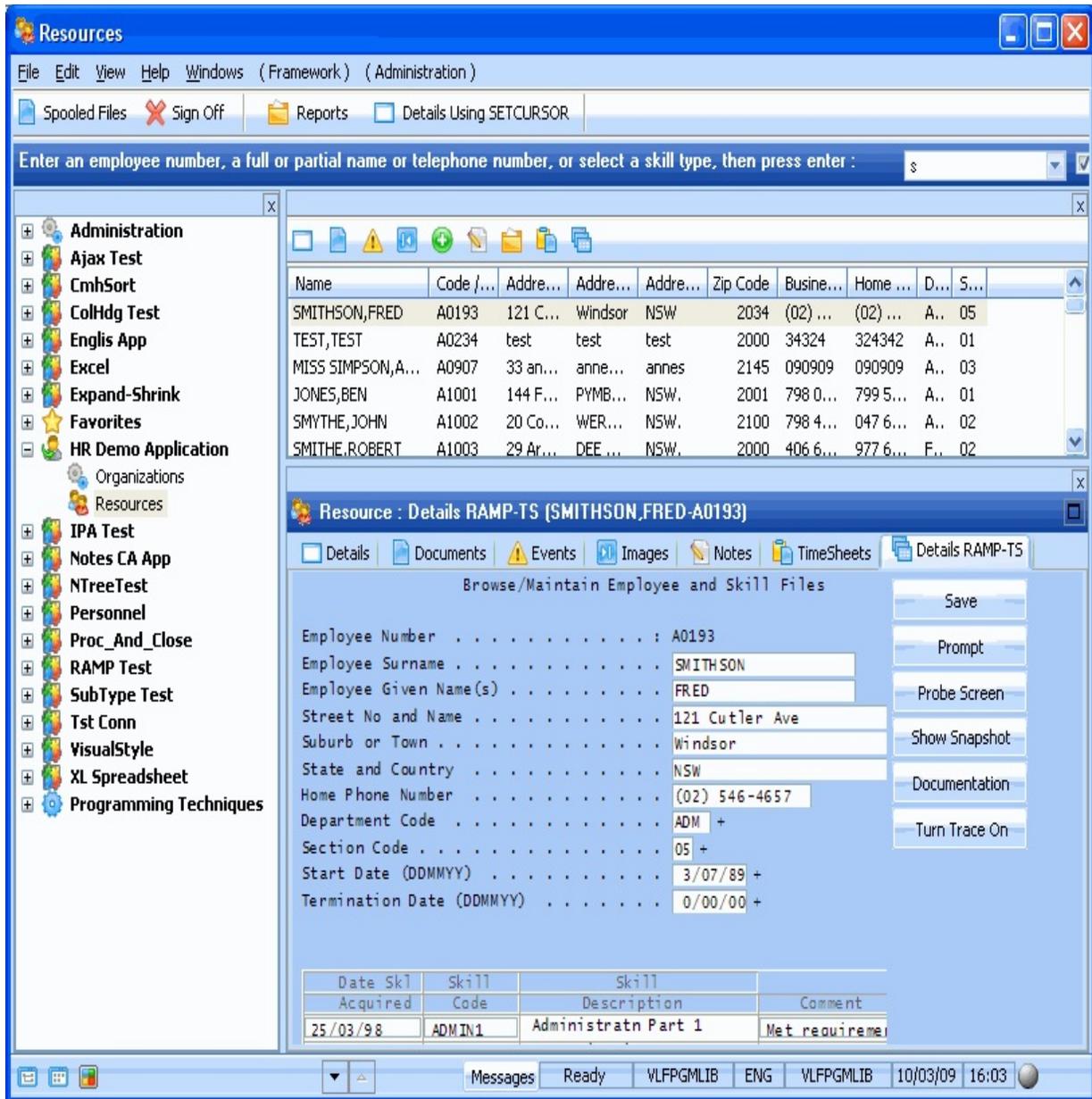
8. Close DF_FILT9.

9. Start the Framework.
10. Start the RAMP Tools.
11. Locate the script associated with the FindEmployee screen in the screens and scripts list.
11. Change the SETVALUE statement to:

```
var wBusinessObject = objBusinessObject.uUserObjectType;  
    if (wBusinessObject == 'EMPLOYEES')  
SETVALUE("txtEmpno",objListManager.AKey1[0]);  
    if (wBusinessObject == 'DEM_ORG_SEC_EMP')  
SETVALUE("txtEmpno",objListManager.AKey3[0]);
```

This statement sets the AKey value according to the name of the business object that is invoking the screen.

12. Commit changes and do a partial save.
13. Display the Framework.
14. Select Resources in the HR Demo Application and fill the instance list.
15. Display the Details RAMP-TS command handler:



The HR Demo Application now consists of a mix of modernized 5250 screens and Framework components.

16. Verify that the Details command tab on the Employees business object is also still functional.

Summary

Important Observations

You can easily mix RAMPed 5250 screens with command handlers created with Visual LANSA

The filter determines which AKey values need to be used to link your screens with the instance list.

What You Should Know

How to combine RAMP screens and Visual LANSA command handlers

How the AKey values are established.

RAMP-TS015: Understanding and Handling Screen Variations

In this tutorial you will learn to understand concepts behind screen signatures and screen identification.

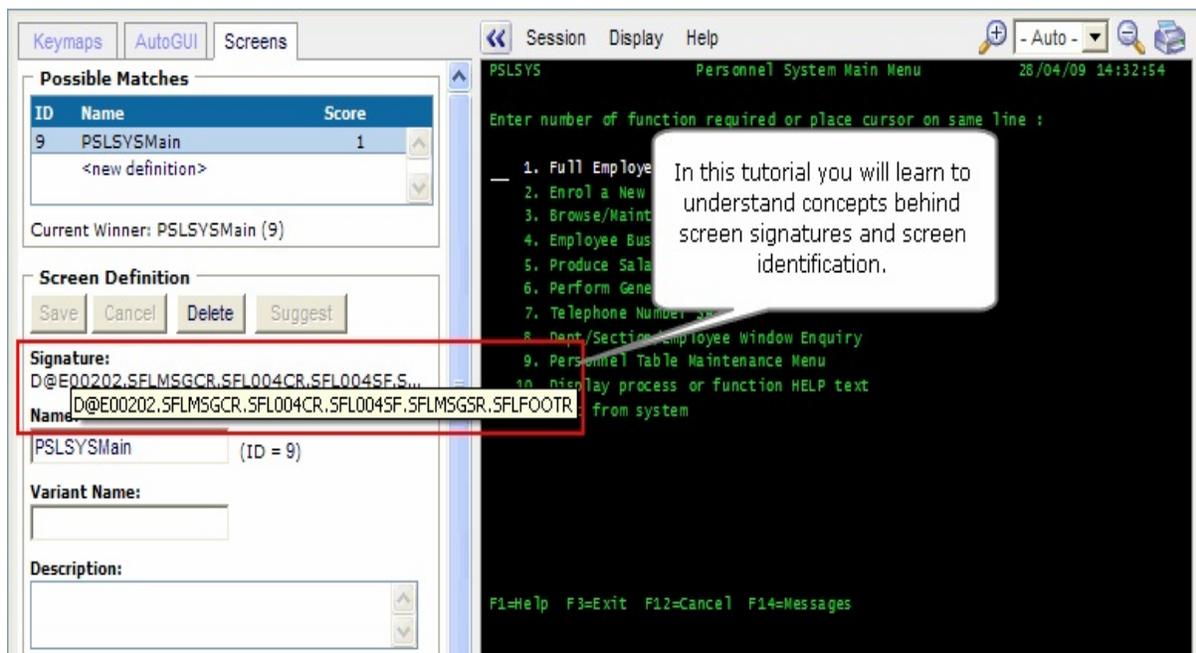
Objectives

Understand what screen signatures are.

Learn how to create a single screen definition for two screens with different signatures

Learn how to uniquely identify screens that share the signature with other screens.

Learn how to use screen name Variants.



To achieve this objective, you will complete the following steps:

[RAMP-TS015 Step 1. Assigning the Same Name to Two Screen Variations](#)

[RAMP-TS015 Step 2. Handling Different Screens with the Same Signature](#)

[RAMP-TS015 Step 3. Creating Screen Variants](#)

[RAMP-TS015 Step 4. Using Screen Variants in the Script](#)

[RAMP-TS015 Step 5. Creating a Set of Screens \(Advanced\)](#)

Before You Begin

In order to complete this tutorial, you must have completed the following:

[RAMP-TS001: Creating a Basic Prototype of the Modernized Application](#)

[RAMP-TS002: Rapidly Modernizing Complete Application Segments](#)

[RAMP-TS003: Creating a Data Filter for Employees](#)

[RAMP-TS004: Naming and Classifying the Employee Screens](#)

[RAMP-TS005: Reviewing Design](#)

[RAMP-TS006: Snapping in a Basic Inquiry Screen](#)

[RAMP-TS007: Snapping in a Data Entry Function](#)

[RAMP-TS008: Changing Inquiry Screen to Update Mode](#)

[RAMP-TS009: Tracing Navigation](#)

[RAMP-TS010: Using Special Field Handlers](#)

[RAMP-TS011: Snapping in Shipped Documents Command Handler](#)

[RAMP-TS012: Snapping in Shipped Notes Command Handler](#)

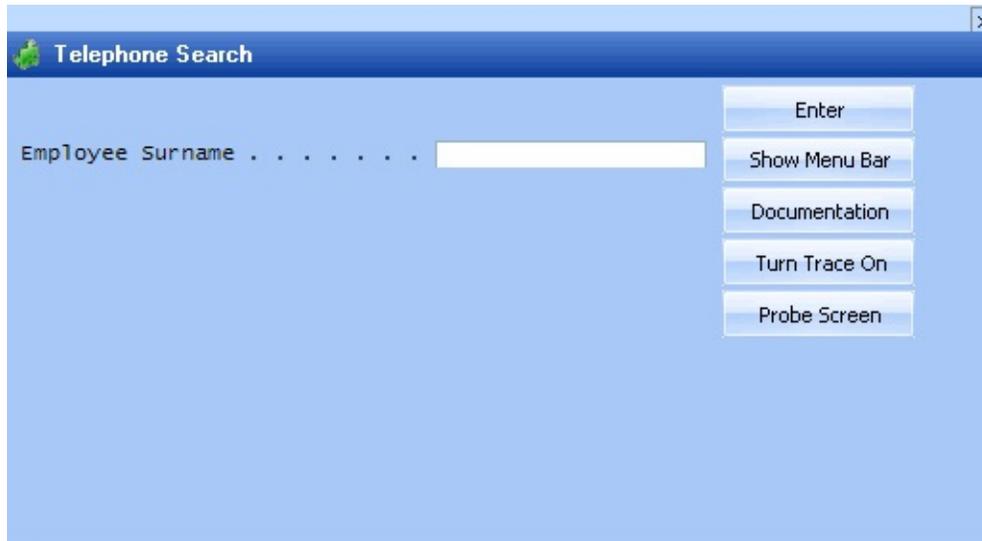
[RAMP-TS013: Sending Instance List Data to Excel](#)

[RAMP-TS014: Snapping RAMP Screens into the HR Demo Application](#)

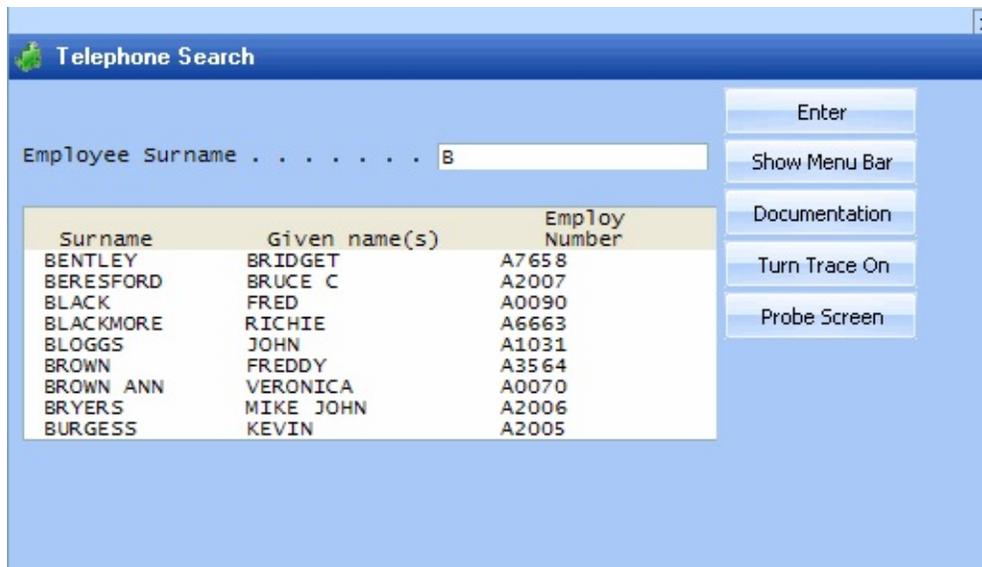
What is a 5250 Screen?

The question "What is a 5250 screen?" is subjective.

Is this 5250 screen...



the same as this 5250 screen?



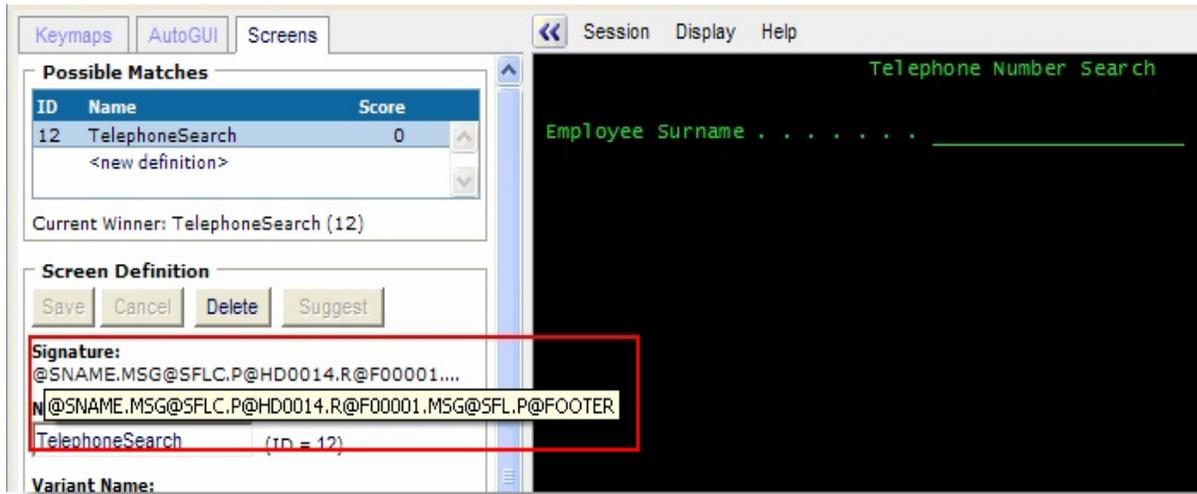
On appearance, you might say "No they are different - you can see that just by looking at them".

However, if you know how the RPG program displaying them works you might say "Yes, these are the same screen" because there is only one point in the program that actually displays a 5250 screen, so logically they must be the same

screen.

Screen Signatures

RAMP-TS assigns a signature to every 5250 screen based on the name of the record formats displayed on the screen. You can see the signature when using RAMP-Tools.



The preceding example screens have different signatures because the second screen displays a subfile control record and some subfile records. This means RAMP-TS will consider them to be different screens.

The same sort of different signature situation can happen on other non-subfile screens.

For example, an order details display may display a record format called ADDINFO (say) that shows addressing details, but it only does this when the delivery address is different to the postal address. This means that what you think is a single screen named OrderDetails (say) actually comes two variations (ie: it has two distinct signatures).

The key to this tutorial is understanding that 5250 screens have different signatures and how you can use these to handle different situations.

Handling the same screen being displayed with different signatures

If RAMP gives different signatures to two screens that you think should actually be the same screen, you can easily resolve this by giving both screens the same screen name.

This means that there will be a single screen script handling both the screen variations.

Sometimes you also apply a variant name to each different screen signature so that the single screen script can tell which screen it is actually handling.

See [RAMP-TS015 Step 1. Assigning the Same Name to Two Screen Variations](#) to learn how to handle this situation.

Handling different screens being displayed with same signature

Sometimes what you consider to be different screens will have the same signature.

Typically this is the case with i5/OS system command screens (all have the signature QDUI132.USRRCD).

You can uniquely identify these screens by selecting additional details on the screen as ID fields. For example, the title of the screen. Once you do this you each screen is assigned a different screen name, and thus has its own unique screen handling script.

See [RAMP-TS015 Step 2. Handling Different Screens with the Same Signature](#)

Handling different screens as group or set of screens

Sometimes a whole set of different screens have very similar behavioral characteristics (for example, code table maintenance programs).

Each screen would have a different signature, but if you assign the same screen name to them all, you will have a single screen script managing them all.

This is productive because a single screen script can handle many different screen variations.

Typically you also assign each different screen a different variant name so that your single screen script can tell which one it is actually handling.

See [RAMP-TS015 Step 5. Creating a Set of Screens \(Advanced\)](#).

RAMP-TS015 Step 1. Assigning the Same Name to Two Screen Variations

In this step you will give the same name to two variations of a screen (with different signatures), one with an empty subfile and one with a subfile with entries.

By defining the two variations as the same screen, there will only be one script to control the navigation to and from the screen.

The function used in this step is the Telephone Search function in the Personnel System, which contains a single REQUEST statement display the screen:

```
* *****
* Process   : PLSYS (Personnel system)
* Function  : SNAME (Search for employee by name)
* *****

* Define all groups, lists and work fields used

FUNCTION OPTIONS(*NOMESSAGES *DEFERWRITE *DBOPTIMISE *DIRECT *LIGHTUSAGE *MLOPTIMISE)
DEFINE FIELD(##SURNAME) REFFLD(##SURNAME)
DEF_LIST NAME(##PSLNAM) FIELDS(##SURNAME ##GIVENAME ##EMPNO ##PHONEBUS) ENTRYS(0000100)
* Program mainline
BEGIN LOOP
  * Request next search name / display previous list
  REQUEST FIELDS(##SURNAME) IDENTIFY(*DESC) ACROSS_SEP(001) BROWSELIST(##PSLNAM)
  CHANGE FIELD(##SURNAME) TO(##SURNAME)
  * Build new list of employees with requested partial name
  CLR_LIST NAMED(##PSLNAM)
  SELECT FIELDS(##PSLNAM) FROM_FILE(PSLMST2) WITH_KEY(##SURNAME) GENERIC(*YES)
  ADD_ENTRY TO_LIST(##PSLNAM)
  ENDSELECT
  CHANGE FIELD(##SURNAME) TO(##SURNAME)
END_LOOP
```

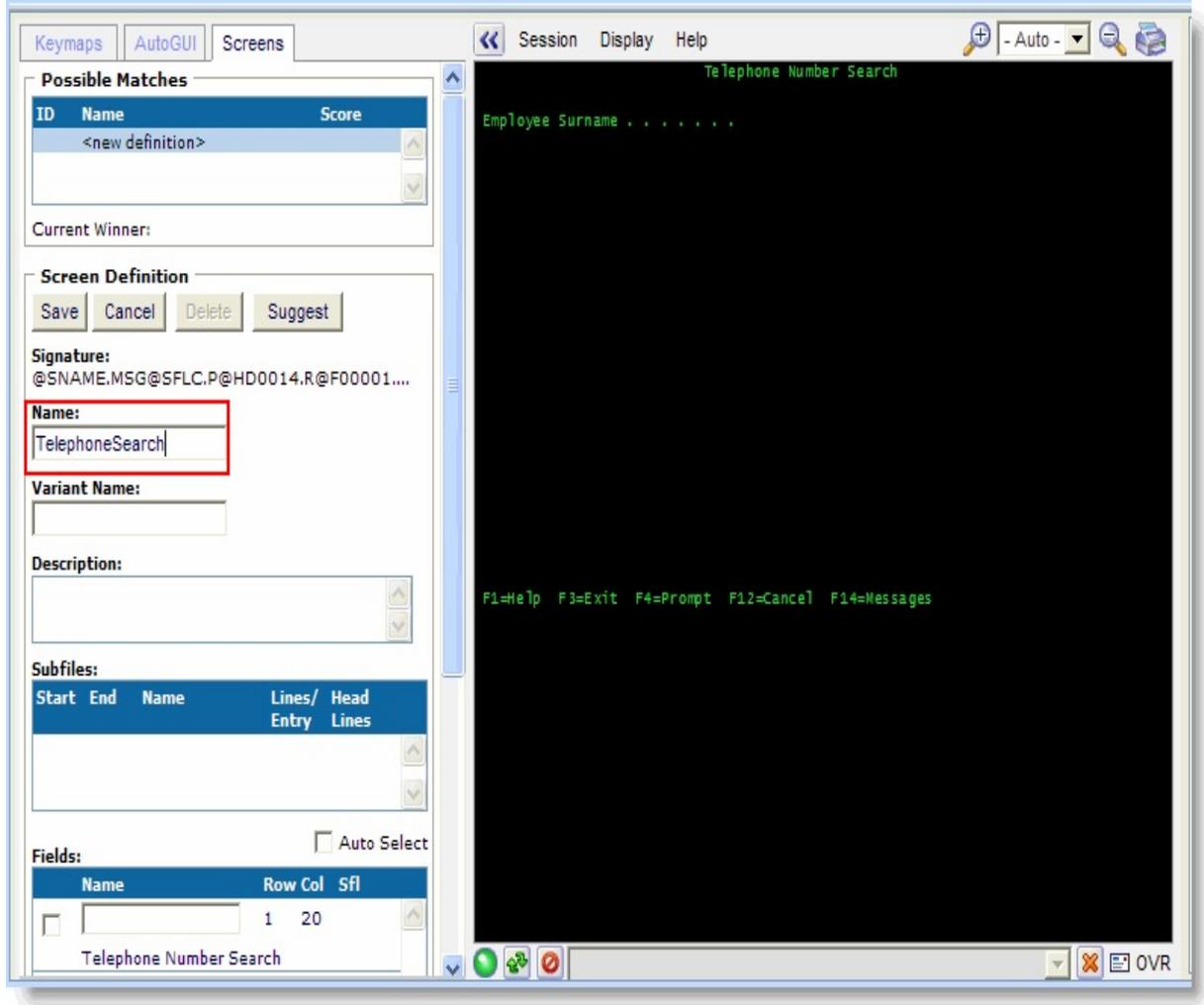
When the screen is initially displayed, it has a different signature to when it is displayed with a list of employees and phone numbers because the record formats are different.

1. In RAMP Tools, start the RAMP-TS 5250 emulator session.
2. Navigate to the Personnel System main menu:

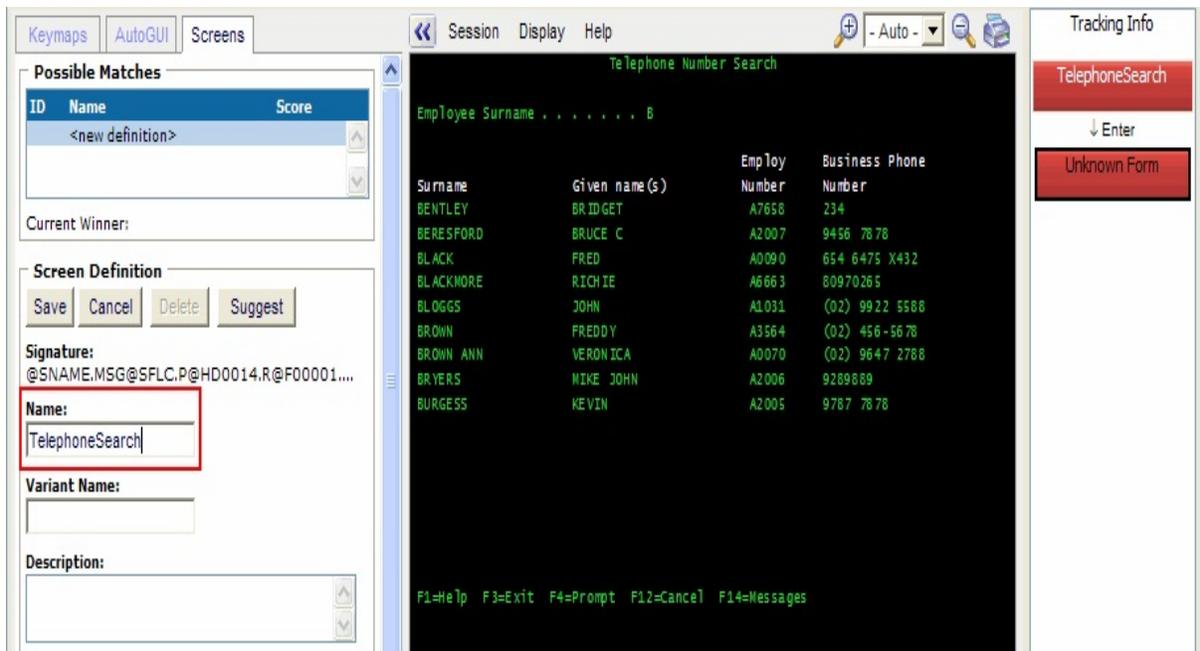
```
LANSA run pslsys partition(dem)
```

3. Select option 7 Telephone Number Search.
4. Name the screen TelephoneSearch and click the Save button on the Screens

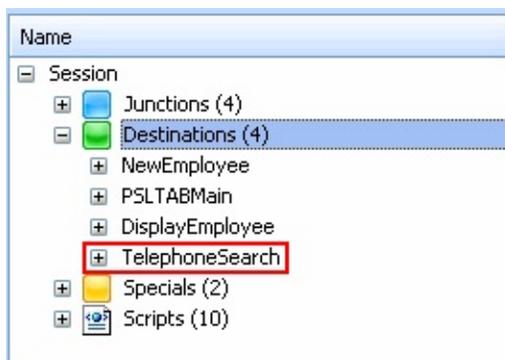
tab:



5. Enter a letter in the Employee Surname field and press Enter. Note that the screen is redisplayed with the subfile and another signature, so RAMP shows the screen as Unknown Form.
6. Give the screen the same name TelephoneSearch and save the name.



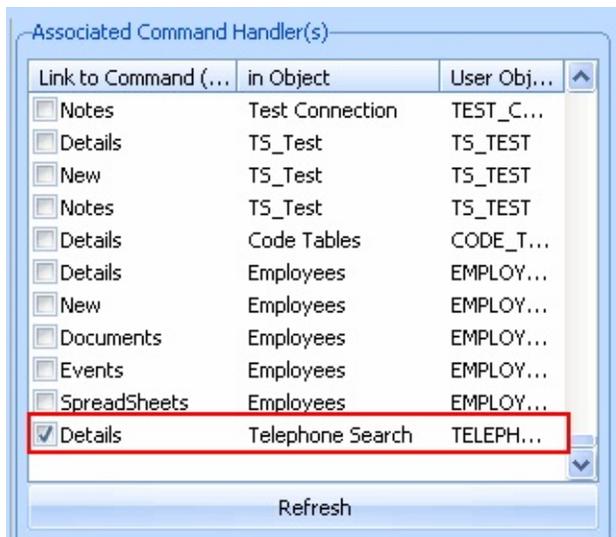
7. Define the TelephoneSearch screen as a destination.
8. Cancel out of the screen and select option 7 again.
9. Enter a letter in the Surname field and press Enter. Notice that both TelephoneSearch screens are now defined as a destination and that they are shown as one screen in the screens and script list:



10. Do a partial save of the RAMP definition.

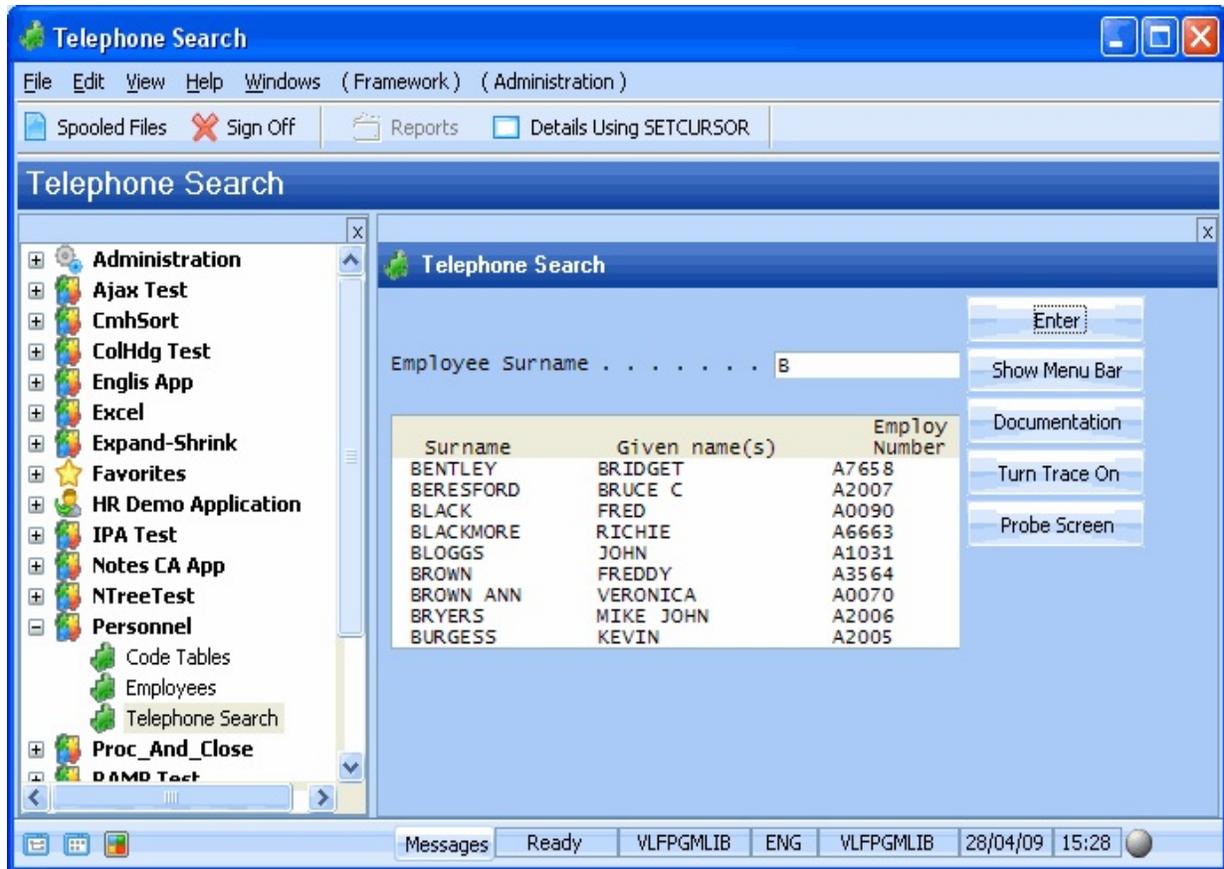
Next you need to create a business object with which to associate the TelephoneSearch screen:

11. In the Framework window, use the Instant Prototyping Assistant to create a new business object Telephone Search, add the Details command to it and then associate it with the Personnel application.
12. When the Telephone Search business object has been created:
 - Open its properties
 - Delete the New Filter created by default
 - Make the command a Business Object Command in the Commands Enabled tab
 - Click the Close button to the message asking to Restart the Framework
 - Close the properties.
13. Display the RAMP Tools window.
14. Select the TelephoneSearch destination screen in the screens and script list and display its details.
15. Refresh the Associated Command Handlers list.
16. Select the Details command handler of the Telephone Search business object as the command handler.



17. Disable all function keys and buttons except Enter.
18. Switch to the Framework window and select the Save and Restart option from the Framework menu.

19. When the Framework window restarts, select the Personnel application and the Telephone Search business object.
20. Test the command handler:



Note that if you want to name any fields on the TelephoneSearch destination, you must do it on both screen variations.

RAMP-TS015 Step 2. Handling Different Screens with the Same Signature

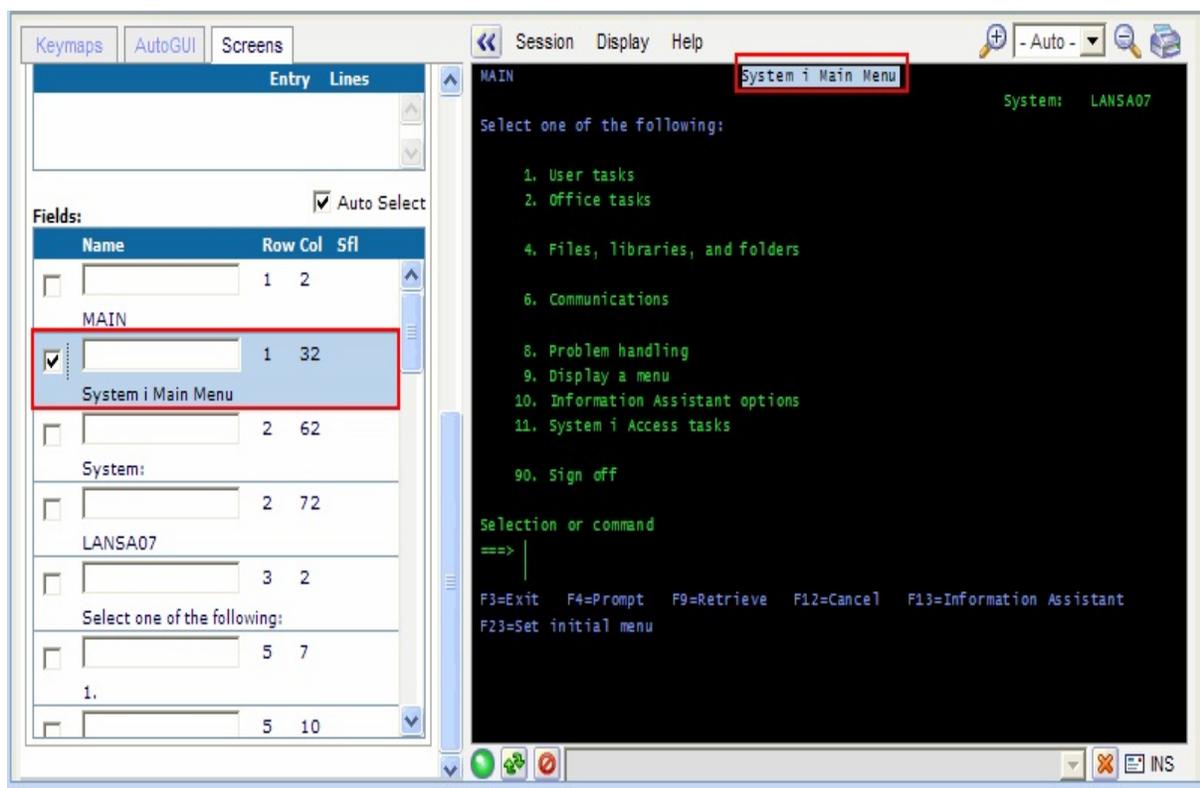
In this step you will give different names to two i5/OS system command screens which have the same signature (all i5/OS system command screens have the same signature), the System i Main Menu and Work with Active Jobs.

You will uniquely identify these screens by selecting additional details on the screen as ID fields.

1. Navigate to the System i Main Menu screen in the RAMP-TS 5250 emulator session.

You have already named the screen `i5OSMainMenu`, but because this screen has the same signature as other i5/OS screens, you need to add more information to its definition to uniquely identify it.

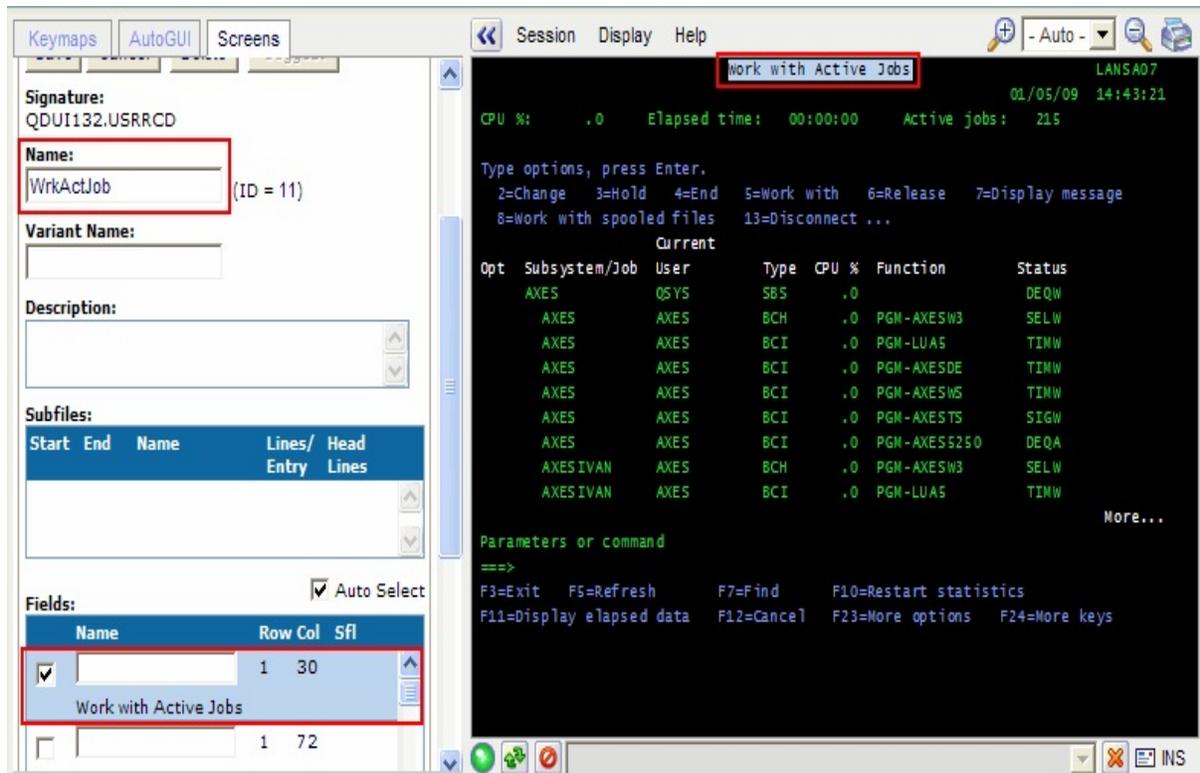
2. Add the title of the screen to the definition by checking the check box in front of it.



3. Click on the Save button on the Screens tab.

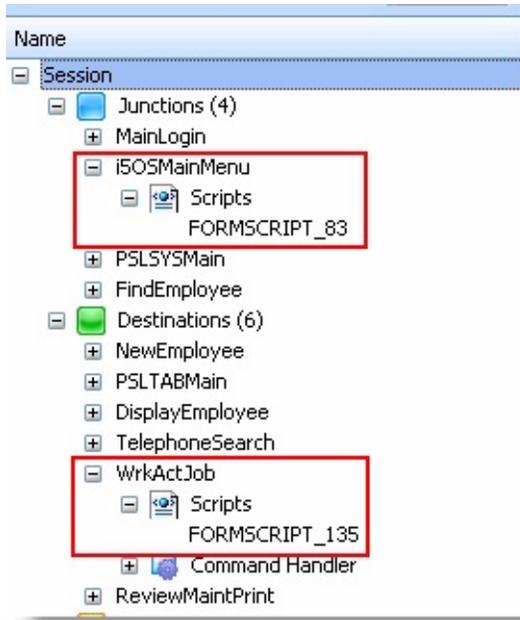
Next name the Work with Active Jobs screen:

4. Navigate to the Work with Active Jobs (WRKACTJOB) screen in the RAMP-TS 5250 emulator session.
5. Add the title area to the screen definition by checking the check box for the title in the Screens tab.
6. Enter WrkActJob as the screen name.



7. Click on the Save button.
8. Classify the WrkActJob screen as a destination.

Note that now the two screens have their own definitions with associated scripts:

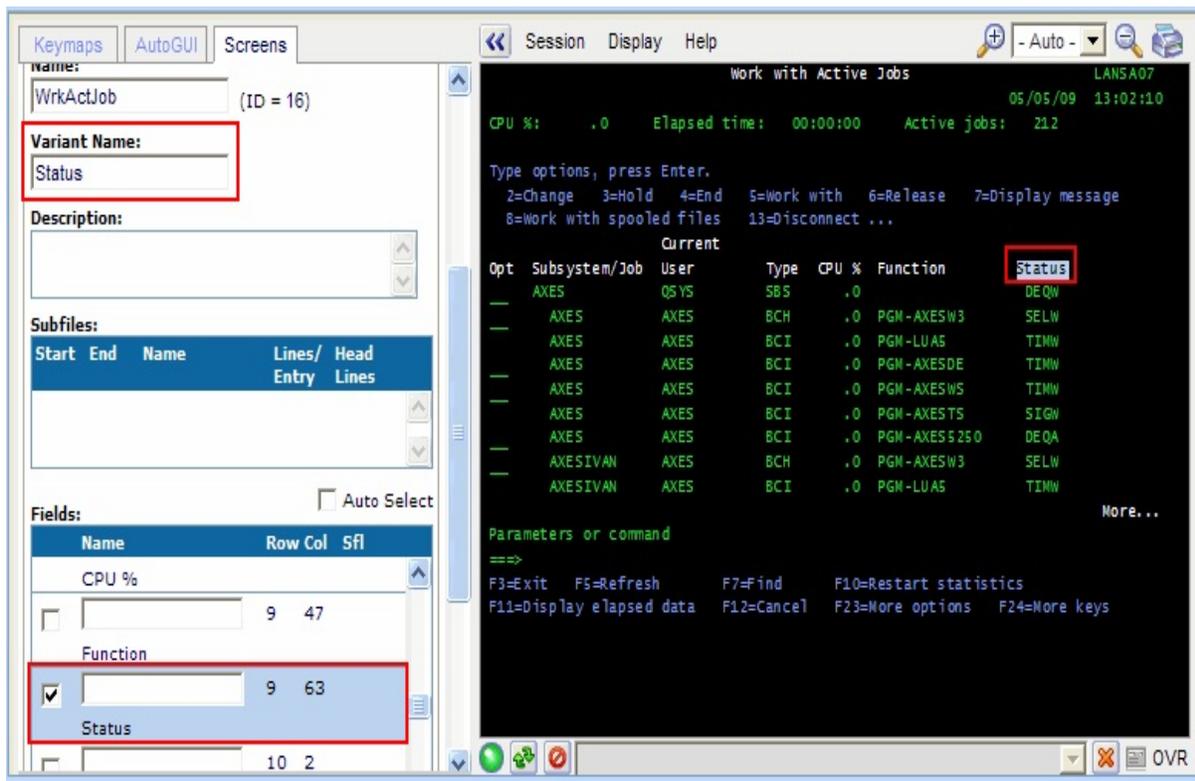


9. Next track the navigation to and from the Work with Active jobs screen.
10. Do a partial save of the RAMP definition.

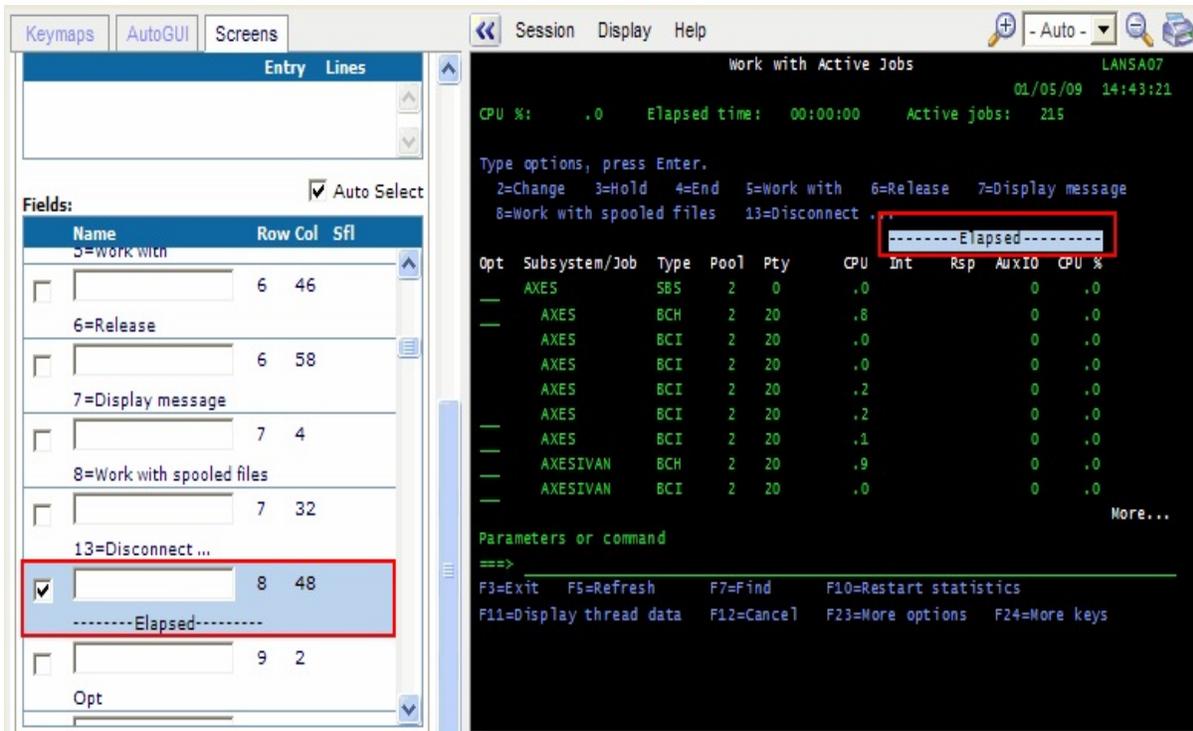
RAMP-TS015 Step 3. Creating Screen Variants

In this step you will identify the different views of the Work with Active Jobs screen as variants, so that your script can handle the view being shown.

1. Enter the Variant name Status for the first view.
2. Add the Status column heading to the screen definition.



3. Click on the Save button on the tab.
4. Press F11 on the Work with Active jobs screen to display the next view.
5. Add the screen's title and the Elapsed column heading to the screen definition.



6. Enter the name of the screen (WrkActJob) and the Variant name Elapsed.
7. Click on the Save button on the Screens tab.
8. Press F11 on the Work with Active jobs screen to display the next view.
9. Add the Threads column heading to screen definition by checking the field corresponding to the title ---- Threads ----.
10. Enter the Variant name Threads.
11. Click on the Save button on the tab.
12. Press F11 to verify the different views have variant names.
13. Do a partial save of the RAMP definition.

Next you will create an application and business object you can use to snap the Work with Active Jobs screen into the Framework.

14. In the Framework window, use the Instant Prototyping Assistant to create a new application System i Server:

Create business object Active Jobs

Associate Active Jobs with a Details command handler

Create application System i Server
Add Active Jobs to the new application.

15. Delete the filter for Active Jobs and make Details a business object command.
16. Save and restart the Framework and start RAMP Tools.
17. In the RAMP Tools window associate the WrkActJob screen with the Details command handler of the Active Jobs business object.
18. Disable the Cancel key and button.
19. Select the Session option SESSION_D to execute the destination screen in a separate session.
20. Do a partial save of the RAMP definition.

RAMP-TS015 Step 4. Using Screen Variants in the Script

In this step you will use the screen variants to set the caption of the button corresponding to the F11 key according to the view shown.

1. In the Framework window, select the System i Server application and the Active Jobs business object to Display the Work with Active Jobs screen.
2. Press F11 to display the different views of the screen.
3. Notice that the Caption of the button corresponding to the F11 key has a static caption Display Elapsed Data.

To set the appropriate caption on the button depending on the view shown, you need to check which variant is being shown and then use the `OVERRIDE_KEY_CAPTION_SCREEN` function to set the button caption:

4. Switch to the RAMP Tools window and display the script for the `WrkActJob` screen.
5. In the `vHandle_ARRIVE` function add an `IF... ELSE IF... ELSE` statement after the `bReturn` variable declaration to check which screen variant is shown and to set the caption of the button:

```
if (this.vLatestVariant == "Status")
{
    OVERRIDE_KEY_CAPTION_SCREEN("WrkActJob", KeyF11, "Show
}

else if (this.vLatestVariant == "Elapsed")
{
    OVERRIDE_KEY_CAPTION_SCREEN("WrkActJob", KeyF11, "Show
}

else
{
    OVERRIDE_KEY_CAPTION_SCREEN("WrkActJob", KeyF11, "Show
}
```

Your script will look like this:

```

{
/* ===== */
/* ===== vHandle_ARRIVE ===== */
/* ===== */
/* Handle arrival at this Destination */
/* oPayload: The payload supplied by the event initiator */
/* oPreviousForm: Reference to previous Form object */

vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
var bReturn = true;

if (this.vLatestVariant == "Status")
{
OVERRIDE_KEY_CAPTION_SCREEN("WrkActJob", KeyF11, "Show Elapsed Data");
}

else if (this.vLatestVariant == "Elapsed")
{
OVERRIDE_KEY_CAPTION_SCREEN("WrkActJob", KeyF11, "Show Thread Data");
}

else
{
OVERRIDE_KEY_CAPTION_SCREEN("WrkActJob", KeyF11, "Show Status");
}

SHOW_CURRENT_FORM(true); /* Show the form in the framework and show VLF buttons */
HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons displayed */
GET_FORM_MESSAGE(22); /* Extract messages and hide the message line */
SETBUSY(false); /* Last thing done - turn off the busy state */

/* <ARRIVE /> - Do not remove or alter this line */
}
}

```

6. Commit the changes and do a partial save of the RAMP definition.
7. Switch to the Framework window and test your changes. The button caption will change depending on the screen variant shown:

Active Jobs

File Edit View Help Windows (Framework) (Administration)

Spooled Files Sign Off Reports Details Using SETCURSOR

Active Jobs

- Administration
- Ajax Test
- CmhSort
- ColHdg Test
- Englis App
- Excel
- Expand-Shrink
- Favorites
- HR Demo Application
- IPA Test
- Notes CA App
- NTreeTest
- Personnel
- Proc_And_Close
- RAMP Test
- SubType Test
- System i Server
 - Active Jobs
- Tst Conn
- VisualStyle
- XL Spreadsheet
- Programming Techniques

Active Job

CPU %: .0 Elapsed time: 00:00:00 Active jobs: 21

Type options, press Enter.

2=Change 3=Hold 4=End 5=work with 6=Release 7=Display
 8=work with spooled files 13=Disconnect ...

Opt	Subsystem/Job	Type	Pool	Pty	CPU	Int	Rsp	AuxIO	-----Elapsed-----
<input type="checkbox"/>	AXES	SBS	2	0	.0			0	
<input type="checkbox"/>	AXES	BCH	2	20	1.1			0	
<input type="checkbox"/>	AXES	BCI	2	20	.0			0	
<input type="checkbox"/>	AXES	BCI	2	20	.0			0	
<input type="checkbox"/>	AXES	BCI	2	20	.3			0	
<input type="checkbox"/>	AXES	BCI	2	20	.3			0	
<input type="checkbox"/>	AXES	BCI	2	20	.2			0	
<input type="checkbox"/>	AXESIVAN	BCH	2	20	.6			0	
<input type="checkbox"/>	AXESIVAN	BCI	2	20	.0			0	

Parameters or command
 ===>

Enter

Refresh

Find

Restart statistics

Show Thread Data

More options

More keys

Show Menu Bar

Documentation

Turn Trace On

Probe Screen

F3=Exit F5=Refresh F7=Find F10=Restart st: Messages Ready VLFPGLIB ENG VLFPGLIB 5/05/09 11:58

RAMP-TS015 Step 5. Creating a Set of Screens (Advanced)

In this tutorial you will create a set of screens with very similar behavioral characteristics (even though each screen has a different signature) by giving them the same screen name. In this way you will have a single screen definition and script managing them all.

The screens used in this tutorial are the screens accessed from the Code Tables business object in the Personnel application in Framework in [RAMP-TS002: Rapidly Modernizing Complete Application Segments](#):

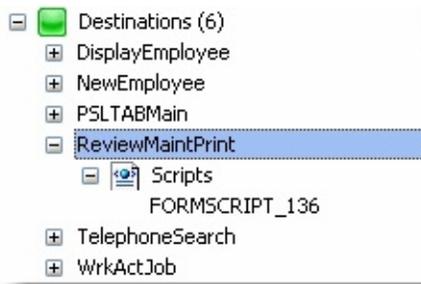
Review/Maintain/Print Department Table

Review/Maintain/Print Section Table

Review/Maintain/Print Skill Table

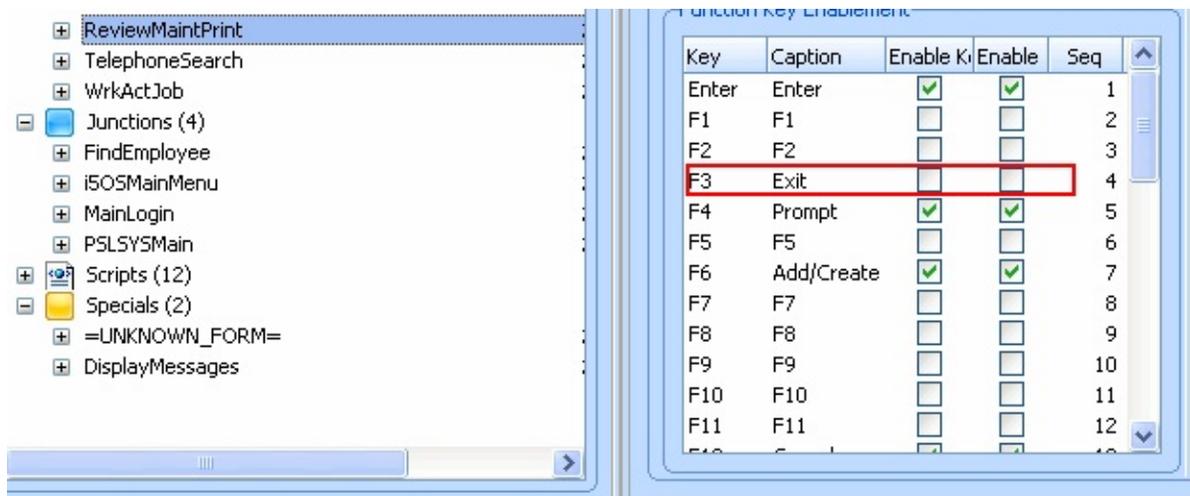
You may want to review these screens first.

1. In the RAMP Tools window ensure that the Auto Update Navigation Scripts option is selected.
2. Use the RAMP-TS 5250 emulator to navigate to the Personnel Table Main Menu (PSLTABMain).
3. Select option 1. Review/Maintain/Print Department Table.
4. Name the Review/Maintain/Print Department Table screen ReviewMaintPrint.
5. Also give it the Variant Name DepartmentTable in case you might want to handle this particular screen in the script. (This is optional, you would normally only give a variant name if you knew you would need it).



Because there is a common screen definition, any changes you make to it affects all the three screens.

16. Disable the Exit key and button.



17. Do a partial save of the RAMP definition.

18. Restart the Framework.

19. Switch to the Framework window and choose the Code Tables business object created in an earlier tutorial.

20. Select the Review/Maintain/Print options and verify that the Exit key and button are not enabled in any of the three screens:

Code Tables [Minimize] [Maximize] [Close]

File Edit View Help Windows (Framework) (Administration)

Spooled Files Sign Off Reports Details Using SETCURSOR

Code Tables

- Administration
- Ajax Test
- CmhSort
- ColHdg Test
- Englis App
- Excel
- Expand-Shrink
- Favorites
- HR Demo Application
- IPA Test
- Notes CA App
- NTreeTest
- Personnel
 - Code Tables
 - Employees
 - Telephone Search
- Proc_And_Close
- RAMP Test
- SubType Test
- System i Server

Code Table

Sel	Dept Code	Section Code	Section Description
<input type="checkbox"/>	ADM	01	INTERNAL ADMIN SRV2
<input type="checkbox"/>	ADM	02	PURCHASING SECTION
<input type="checkbox"/>	ADM	03	ACCOUNTING SECTION
<input type="checkbox"/>	ADM	04	SALES & MARKETING
<input type="checkbox"/>	ADM	05	MAINTENANCE
<input type="checkbox"/>	ADM	06	PERSONNEL SECTION
<input type="checkbox"/>	ADM	09	VEHICLE MAINTENANCE
<input type="checkbox"/>	AUD	01	ADMINISTRATION
<input type="checkbox"/>	AUD	02	PURCHASING
<input type="checkbox"/>	AUD	03	ACCOUNTING
<input type="checkbox"/>	FLT	01	ADMINISTRATION
<input type="checkbox"/>	FLT	02	PURCHASING
<input type="checkbox"/>	FLT	03	ACCOUNTING
<input type="checkbox"/>	GAC	FC	FINANCIAL CONTROL
<input type="checkbox"/>	GAC	01	ADMINISTRATION
<input type="checkbox"/>	GAC	02	PURCHASING

Enter

Prompt

Add/Create

Cancel

Messages

Print

Show Menu Bar

Documentation

Turn Trace On

Probe Screen

Select section to review/char Messages Ready VLFPGLIB ENG VLFPGLIB 6/05/09 14:30

Summary

Important Observations

RAMP assigns signatures to screens based on the record formats on the screen

You can give two or more screens with different signatures the same name.

When you do this, the screens are defined in a single screen definition and there is only a one script that handles them.

You have the option of identifying different variations of a screen if you want to handle them in a different way in the script.

Minimizing the number of screen definitions and scripts you create and maintain can be very productive.

Conversely, you can give different names to screens with the same signature. In this case you will need to add a screen element that uniquely identifies the screens to the screen definitions. This is something you would typically do for the i5/OS system command screens which all have the same signature.

-

What You Should Know

What screen signatures are.

How to give screens with different signatures the same name.

How to identify screens that share the signature with another screen.

How to create and use screen name variants.

Advanced Tutorials

These advanced tutorials demonstrate RAMP-TS concepts and techniques.

[RAMP-TSAD01: Using Buttons to Add Value to an Application](#)

[RAMP-TSAD02: RAMP-TS Event Handling Basics](#)

[RAMP-TSAD03: Special Field Handling](#)

[RAMP-TSAD04: Redesigning the Screen Using aXes](#)

[RAMP-TSAD05: Using SHARED Properties and Functions](#)

[RAMP-TSAD06: Handling Multiple Screens on Multiple Tabs](#)

[RAMP-TSAD07: Handling Multiple Screens on a Single Tab](#)

[RAMP-TSAD08: Screen Wrapper Basics](#)

[RAMP-TSAD09: Screen Wrapper with a Subfile](#)

RAMP-TSAD01: Using Buttons to Add Value to an Application

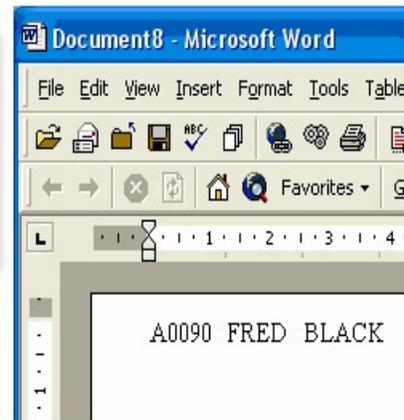
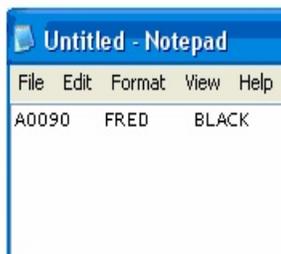
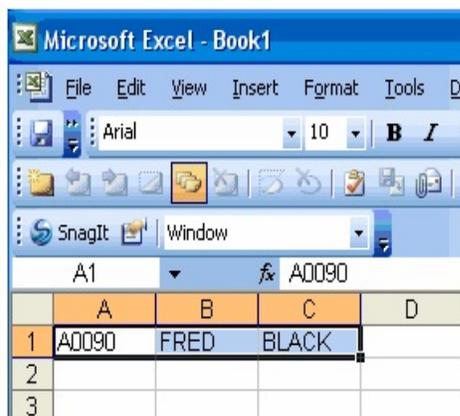
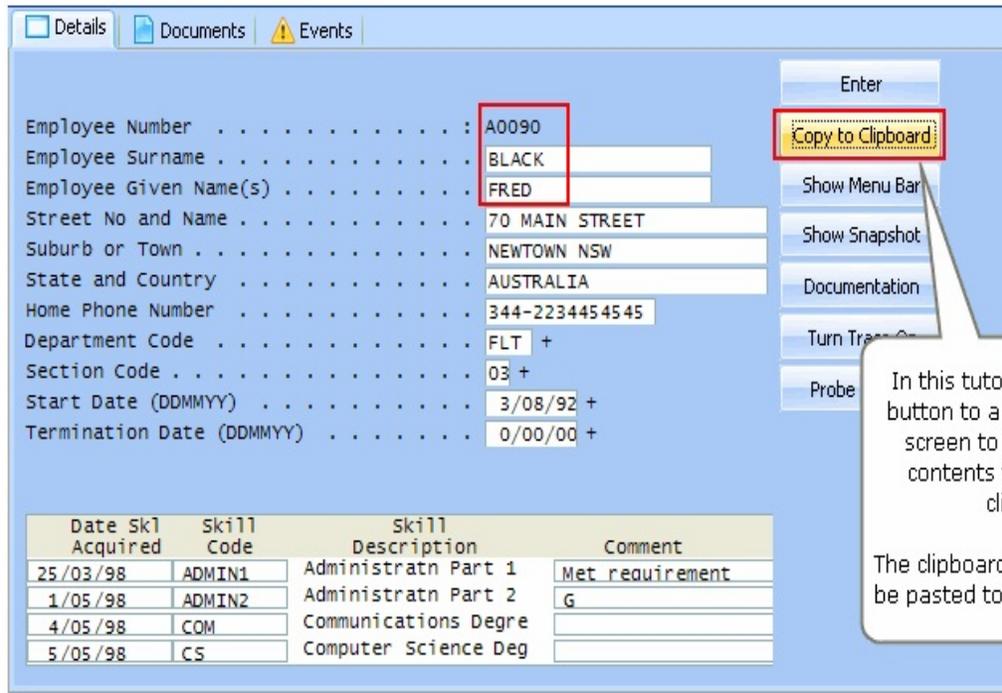
This tutorial will demonstrate how to add useful functionality to a modernized 5250 application by using framework buttons.

Objectives

Learn how to add functionality to your modernized screen using buttons

Understand that clicking a button (or pressing a function key) on a modernized screen does NOT have to interact with the 5250 application. You can capture button or function key on the client to add functionality.

Learn how to copy data from a RAMP command handler to the Windows clipboard



To achieve this objective, you will complete the following steps:

Read [About Buttons](#)

[RAMP-TSAD01 Step 1. Enable Framework Buttons](#)

[RAMP-TSAD01 Step 2. Name Fields to Be Copied on the DisplayEmployee Screen](#)

[RAMP-TSAD01 Step 3. Add a Function to the Script for the DisplayEmployee Screen](#)

[RAMP-TSAD01 Step 4. Call the Function in the ButtonClick Function](#)

[RAMP-TSAD01 Step 5. Test the Buttons](#)

Summary

-

Before You Begin

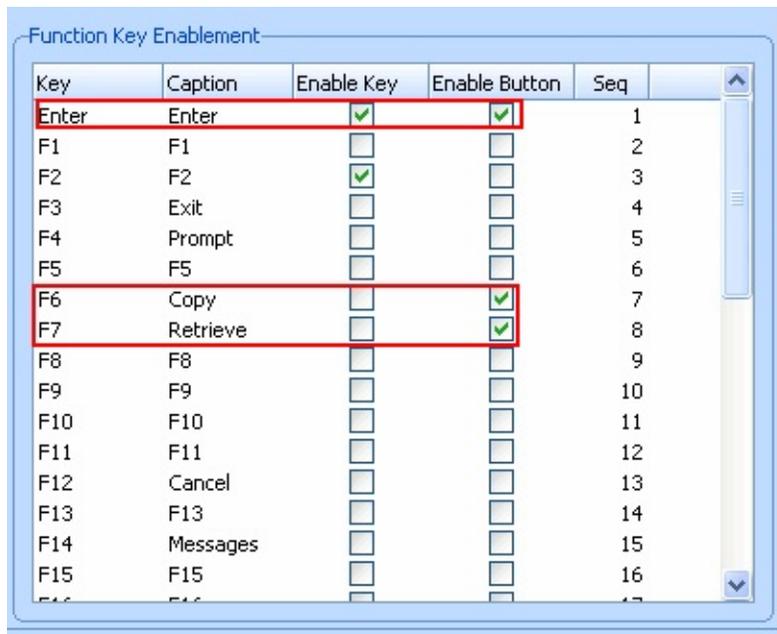
In order to complete this tutorial, you must have completed the core tutorials RAMP-TS001 - RAMP-TS015..

About Buttons

There are two kinds of buttons in a RAMP application, Framework buttons and 5250 buttons.

Framework Buttons

Framework buttons are set for destination screens in the screen's RAMP definition:



The screenshot shows a dialog box titled "Function Key Enablement" with a table of function keys. The table has five columns: "Key", "Caption", "Enable Key", "Enable Button", and "Seq". The "Enable Key" and "Enable Button" columns contain checkboxes. The rows for "Enter", "F6", and "F7" are highlighted with a red border, and their "Enable Button" checkboxes are checked with green checkmarks.

Key	Caption	Enable Key	Enable Button	Seq
Enter	Enter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2
F2	F2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3
F3	Exit	<input type="checkbox"/>	<input type="checkbox"/>	4
F4	Prompt	<input type="checkbox"/>	<input type="checkbox"/>	5
F5	F5	<input type="checkbox"/>	<input type="checkbox"/>	6
F6	Copy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	7
F7	Retrieve	<input type="checkbox"/>	<input checked="" type="checkbox"/>	8
F8	F8	<input type="checkbox"/>	<input type="checkbox"/>	9
F9	F9	<input type="checkbox"/>	<input type="checkbox"/>	10
F10	F10	<input type="checkbox"/>	<input type="checkbox"/>	11
F11	F11	<input type="checkbox"/>	<input type="checkbox"/>	12
F12	Cancel	<input type="checkbox"/>	<input type="checkbox"/>	13
F13	F13	<input type="checkbox"/>	<input type="checkbox"/>	14
F14	Messages	<input type="checkbox"/>	<input type="checkbox"/>	15
F15	F15	<input type="checkbox"/>	<input type="checkbox"/>	16

Employee Number : A0070
Employee Surname BROWN ANN
Employee Given Name(s) VERONICA
Street No and Name 12 Railway Street
Suburb or Town Baulkham Hills
State and Country NSW Australia
Home Phone Number TRN
Department Code INF +
Section Code DV +
Start Date (DDMMYY) 28/01/90 +
Termination Date (DDMMYY) 0/00/00 +

Date Skl Acquired	Skill Code	Skill Description	Comment
30/06/96	ADMIN1	Administratn Part 1	Met requirement
1/03/98	ADMIN2	Administratn Part 2	
10/12/95	ADVPGM	Advanced Programming	
4/05/98	COM	Communications Degre	

Enter

Copy

Retrieve

You should use Framework buttons on all destination screens. Button display on junction and special screens is irrelevant because these screens are not shown. When executing the Framework as a Designer, design-time buttons are also shown to help with development tasks:

Framework buttons

Employee Number : A0070
Employee Surname BROWN ANN
Employee Given Name(s) VERONICA
Street No and Name 12 Railway Street
Suburb or Town Baulkham Hills
State and Country NSW Australia
Home Phone Number TRN
Department Code INF +
Section Code DV +
Start Date (DDMMYY) 28/01/90 +
Termination Date (DDMMYY) 0/00/00 +

Design-time buttons

Enter

Copy

Retrieve

Show Menu Bar

Show Snapshot

Documentation

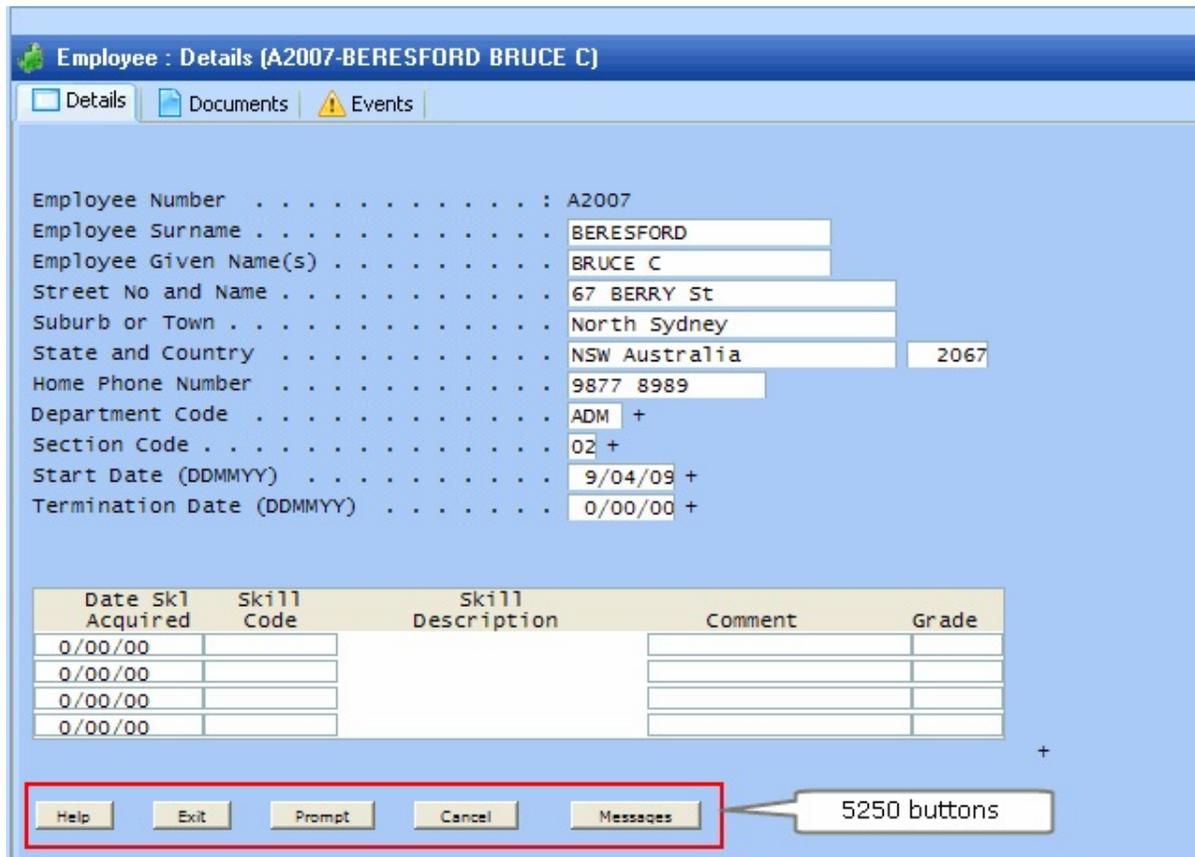
Turn Trace On

Probe Screen

Date Skl Acquired	Skill Code	Skill Description	Comment
30/06/96	ADMIN1	Administratn Part 1	Met requirement
1/03/98	ADMIN2	Administratn Part 2	
10/12/95	ADVPGM	Advanced Programming	
4/05/98	COM	Communications Degre	

5250 Buttons

5250 buttons are the buttons which are part of the modernized 5250 application:



By default on all destination screens the 5250 buttons are hidden. The display of the 5250 buttons is controlled by the `SHOW_5250_BUTTONS()` and `HIDE_5250_BUTTONS()` functions:

```

t
/* ===== */
/* ===== vHandle_ARRIVE ===== */
/* ===== */
/* Handle arrival at this Destination */
/* oPayload: The payload supplied by the event initiator */
/* oPreviousForm: Reference to previous object Form*/

vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;

    if (CHECK_FIELD_EXISTS("txtsurname"))
    {
        SHOW_CURRENT_FORM(true); /* Show the form in the framework and show
        GET_FORM_MESSAGE(22); /* Extract messages and hide the message ]
        HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons displayed
        SETBUSY(false); /* Last thing done - turn off the busy sta
    }
    else
    {
        SENDKEY(KeyF21);
    }
}

```

By default the 5250 buttons are shown in unknown and undefined screens. This behavior is controlled by the special Unknown form definition:

The screenshot displays the RAMP Tools interface for a 'Default Session (RAMP-TS)'. The left pane shows a tree view of the session structure, with 'Specials (2)' expanded to show '=UNKNOWN_FORM=' and 'FORMSCRIPT_77'. The right pane shows the details for 'FORMSCRIPT_77', including its role, script name, caption, and a code editor with the following code:

```

Role : Unknown operation using and
Script : FORMSCRIPT_77
Caption : =UNKNOWN_FORM= - Handle an unknown form
Grouping :

vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;

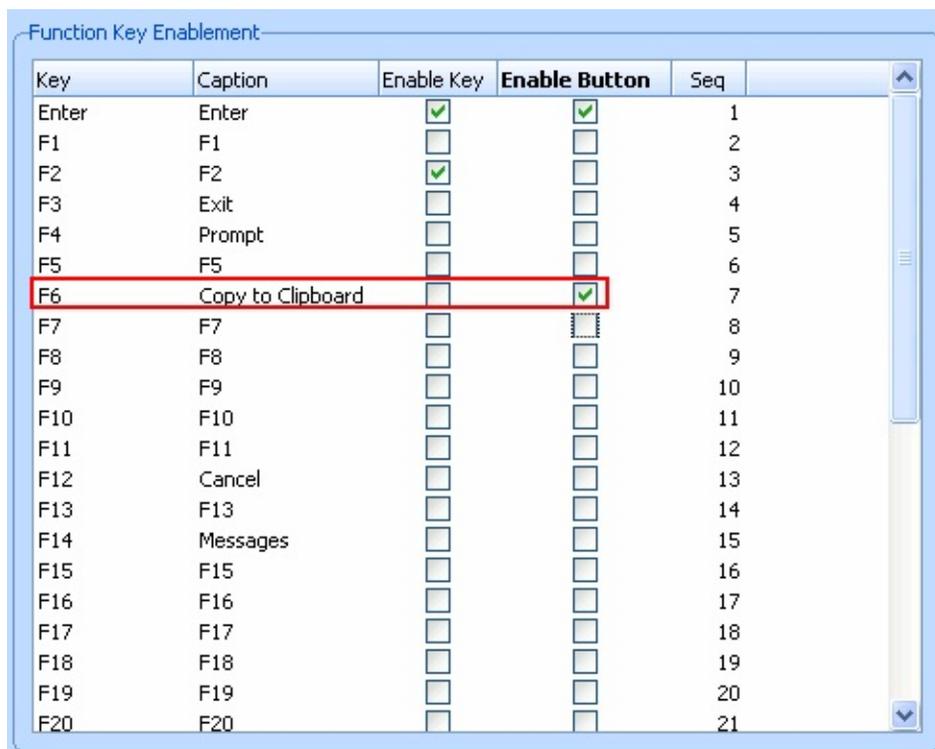
    SHOW_CURRENT_FORM(false);
    SHOW_5250_BUTTONS();
    SETBUSY(false);
}

```


RAMP-TSAD01 Step 1. Enable Framework Buttons

In this step you will add a Copy to Clipboard button to the DisplayEmployee screen.

1. In the RAMP Tools window, open the details of the DisplayEmployee destination screen
2. Check the Enable Button column for function key F6 (you could use any unused key/button).
3. Change the caption to Copy to Clipboard.



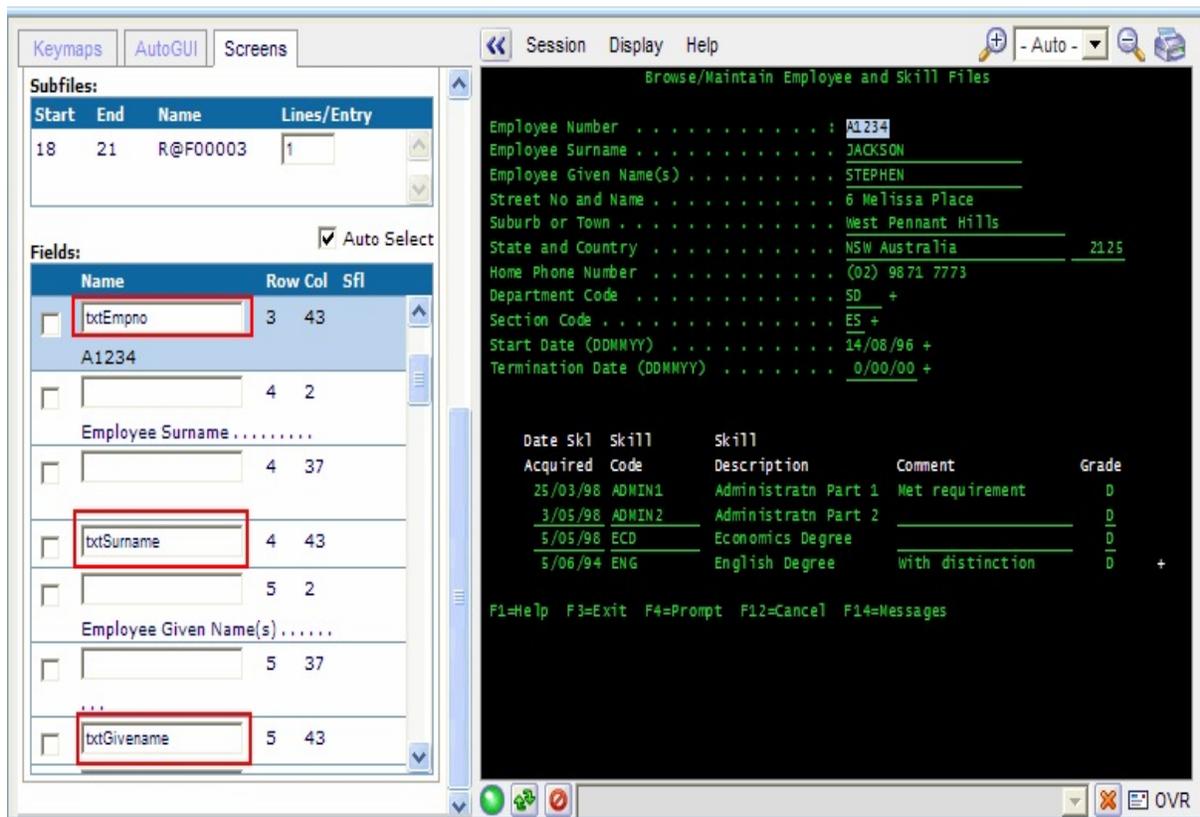
Key	Caption	Enable Key	Enable Button	Seq
Enter	Enter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2
F2	F2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3
F3	Exit	<input type="checkbox"/>	<input type="checkbox"/>	4
F4	Prompt	<input type="checkbox"/>	<input type="checkbox"/>	5
F5	F5	<input type="checkbox"/>	<input type="checkbox"/>	6
F6	Copy to Clipboard	<input type="checkbox"/>	<input checked="" type="checkbox"/>	7
F7	F7	<input type="checkbox"/>	<input type="checkbox"/>	8
F8	F8	<input type="checkbox"/>	<input type="checkbox"/>	9
F9	F9	<input type="checkbox"/>	<input type="checkbox"/>	10
F10	F10	<input type="checkbox"/>	<input type="checkbox"/>	11
F11	F11	<input type="checkbox"/>	<input type="checkbox"/>	12
F12	Cancel	<input type="checkbox"/>	<input type="checkbox"/>	13
F13	F13	<input type="checkbox"/>	<input type="checkbox"/>	14
F14	Messages	<input type="checkbox"/>	<input type="checkbox"/>	15
F15	F15	<input type="checkbox"/>	<input type="checkbox"/>	16
F16	F16	<input type="checkbox"/>	<input type="checkbox"/>	17
F17	F17	<input type="checkbox"/>	<input type="checkbox"/>	18
F18	F18	<input type="checkbox"/>	<input type="checkbox"/>	19
F19	F19	<input type="checkbox"/>	<input type="checkbox"/>	20
F20	F20	<input type="checkbox"/>	<input type="checkbox"/>	21

A button with this caption will appear on the destination screen but the keystroke for F6 will not be sent to the 5250 application because the key has not been enabled. Note also that even if the F6 key had been enabled here, but was not present in the 5250 screen, it would have no effect in the 5250 application.

RAMP-TSAD01 Step 2. Name Fields to Be Copied on the DisplayEmployee Screen

In this step you will name the fields to be copied on the DisplayEmployee screen.

1. Start a RAMP-TS emulator session in RAMP Tools.
2. Ensure that the Auto Update Navigation Scripts check box is not checked.
3. Navigate to the DisplayEmployee screen and press F21 to put it in edit mode.
4. Name the Employee number and Given Name fields txtEmpno and txtGivenname. The Surname field should have been named txtSurname in a previous tutorial:



Bear in mind that the field names are case sensitive.

5. Click on the Save button on the Screens tab.

RAMP-TSAD01 Step 3. Add a Function to the Script for the DisplayEmployee Screen

In this step you will add a function in the script for DisplayEmployee to copy the contents of the Employee, Givename and Surname fields to the Windows clipboard.

1. Locate the script for the DisplayEmployee destination screen.
2. Copy and paste the following function to the script after the NavigateTo function, before the //<SYSINFO> block:

```
/* =====  
/* ===== uCopyEmpDetails ===== */  
/* =====  
/* Copies Employee Details to the Windows Clipboard      */  
  
uCopyEmpDetails : function () {  
  
/* Get details from 5250 screen */  
var TAB_Char = "\x09" ;  
var End_Of_Line_Char = "\x0D\x0A" ;  
  
var strEmpno = GETVALUE("txtEmpno");  
var strGName = GETVALUE("txtGivename");  
var strSName = GETVALUE("txtSurname");  
  
/* Write details to clipboard */  
var MyString = "";  
  
MyString = strEmpno + TAB_Char + strGName + TAB_Char + strSName +  
COPYTOCLIPBOARD(MyString);  
  
/* Issue a message */  
MESSAGE("Details for employee ", strEmpno," sent to the clipboard");  
  
} ,
```

3. Use the Commit Changes button to commit the changes to the script.

RAMP-TSAD01 Step 4. Call the Function in the ButtonClick Function

In this step add code to call the Copy to Clipboard function from the ButtonClick function.

1. Add a case statement for F6 key (KeyF6) in the Switch command of the ButtonClick function of the DisplayEmployee script.
2. In the statement for the F6 key add a call to the uCopyEmpDetails function:

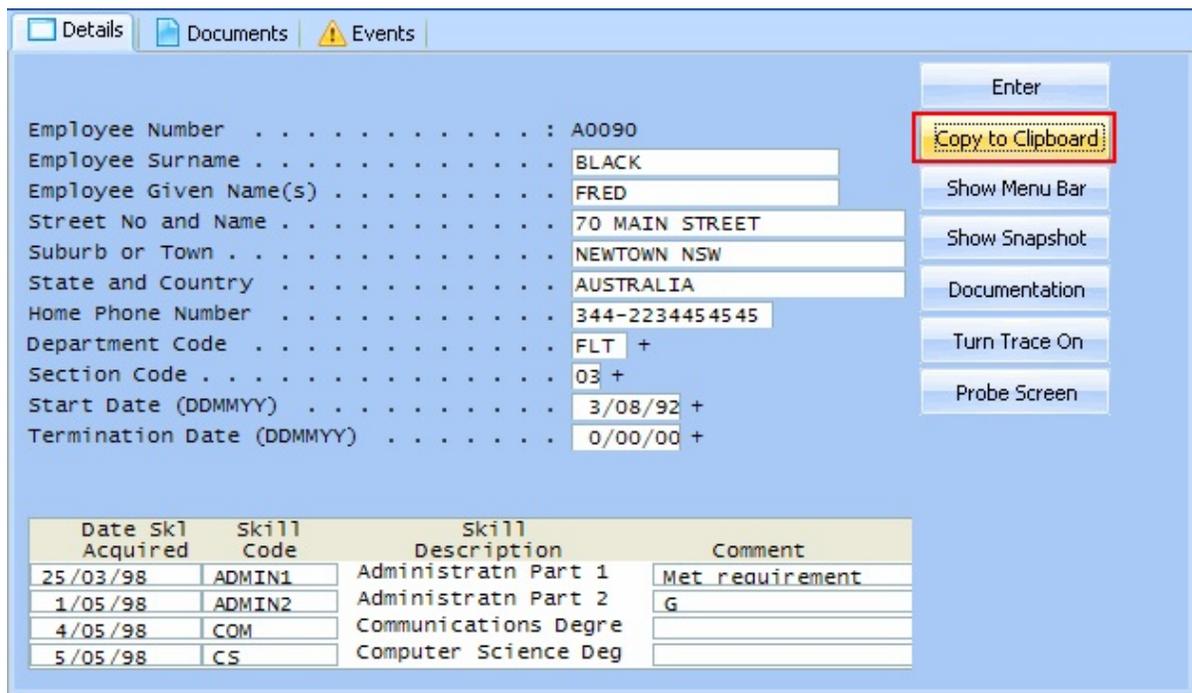
```
case KeyF6:  
    /* Call copy function */  
    this.uCopyEmpDetails();  
    break;
```

3. Use the Commit Changes button to commit the changes to the script and then do a partial save.

RAMP-TSAD01 Step 5. Test the Buttons

In this step you will test the Copy to Clipboard button.

1. In the Framework window, locate the Personnel Application.
2. Select the Employees business object and use the filter to fill the instance list.
3. Select an employee to display its details.
4. Click on the Copy to Clipboard button.



The screenshot shows a software interface with a light blue background. At the top, there are three tabs: 'Details' (selected), 'Documents', and 'Events'. Below the tabs, there is a list of employee details for employee A0090. The details are as follows:

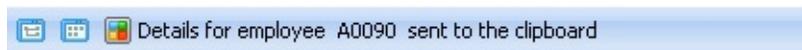
Employee Number	A0090
Employee Surname	BLACK
Employee Given Name(s)	FRED
Street No and Name	70 MAIN STREET
Suburb or Town	NEWTOWN NSW
State and Country	AUSTRALIA
Home Phone Number	344-2234454545
Department Code	FLT +
Section Code	03 +
Start Date (DDMMYY)	3/08/92 +
Termination Date (DDMMYY)	0/00/00 +

At the bottom of the details section, there is a table of skills:

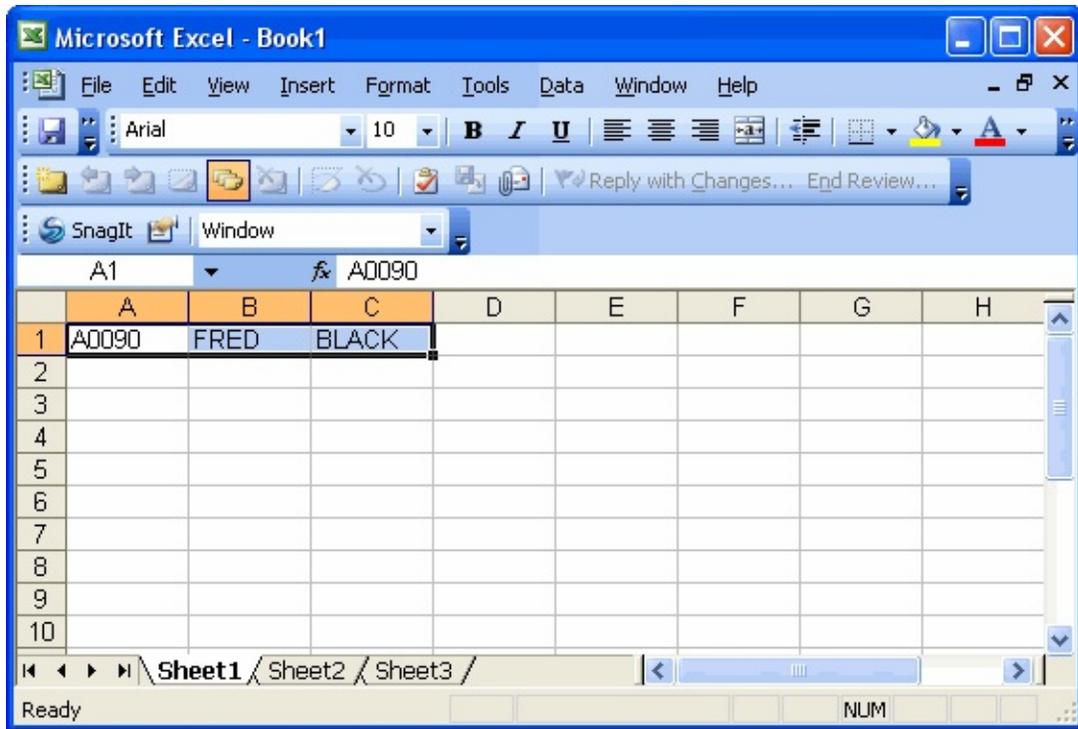
Date Skl Acquired	Skill Code	Skill Description	Comment
25/03/98	ADMIN1	Administratrn Part 1	Met requirement
1/05/98	ADMIN2	Administratrn Part 2	G
4/05/98	COM	Communications Degre	
5/05/98	CS	Computer Science Deg	

On the right side of the interface, there is a vertical stack of buttons: 'Enter', 'Copy to Clipboard' (highlighted with a red box), 'Show Menu Bar', 'Show Snapshot', 'Documentation', 'Turn Trace On', and 'Probe Screen'.

Notice the message indicating the employee details have been copied to the clipboard.



5. Start another application to which you can paste the contents of the clipboard, for example MS Word or Excel.
6. Paste in the employee details (Ctrl + V):



Summary

Important Observations

A function key with the Enable Button checkbox checked but not the Enable Key checkbox functions as a Framework only button. The action of Framework buttons is handled entirely on the client and no keystrokes are sent to the 5250 application. See [Function Key Enablement](#).

The case statement for a function key controls what happens when the key is used from the keyboard or its button is clicked.

You can use the [SETKEYENABLED Function](#) to dynamically enable and disable buttons and function keys.

You can use the [OVERRIDE_KEY_CAPTION_SCREEN Function](#) or the [OVERRIDE_KEY_CAPTION_ALL Function](#) to dynamically change the text on Framework buttons. The `OVERRIDE_KEY_CAPTION_ALL` function can also be used to set all function key captions to another language in a multilingual application.

What You Should Know

How to add value to a modernized 5250 application using Framework buttons.

How to enable a Framework button and modify the ButtonClick function to provide actions when the button is clicked.

How to add functions to a form script that are available from anywhere inside the form script.

RAMP-TSAD02: RAMP-TS Event Handling Basics

This tutorial demonstrates how a RAMP script may signal an event to Framework component and vice versa.

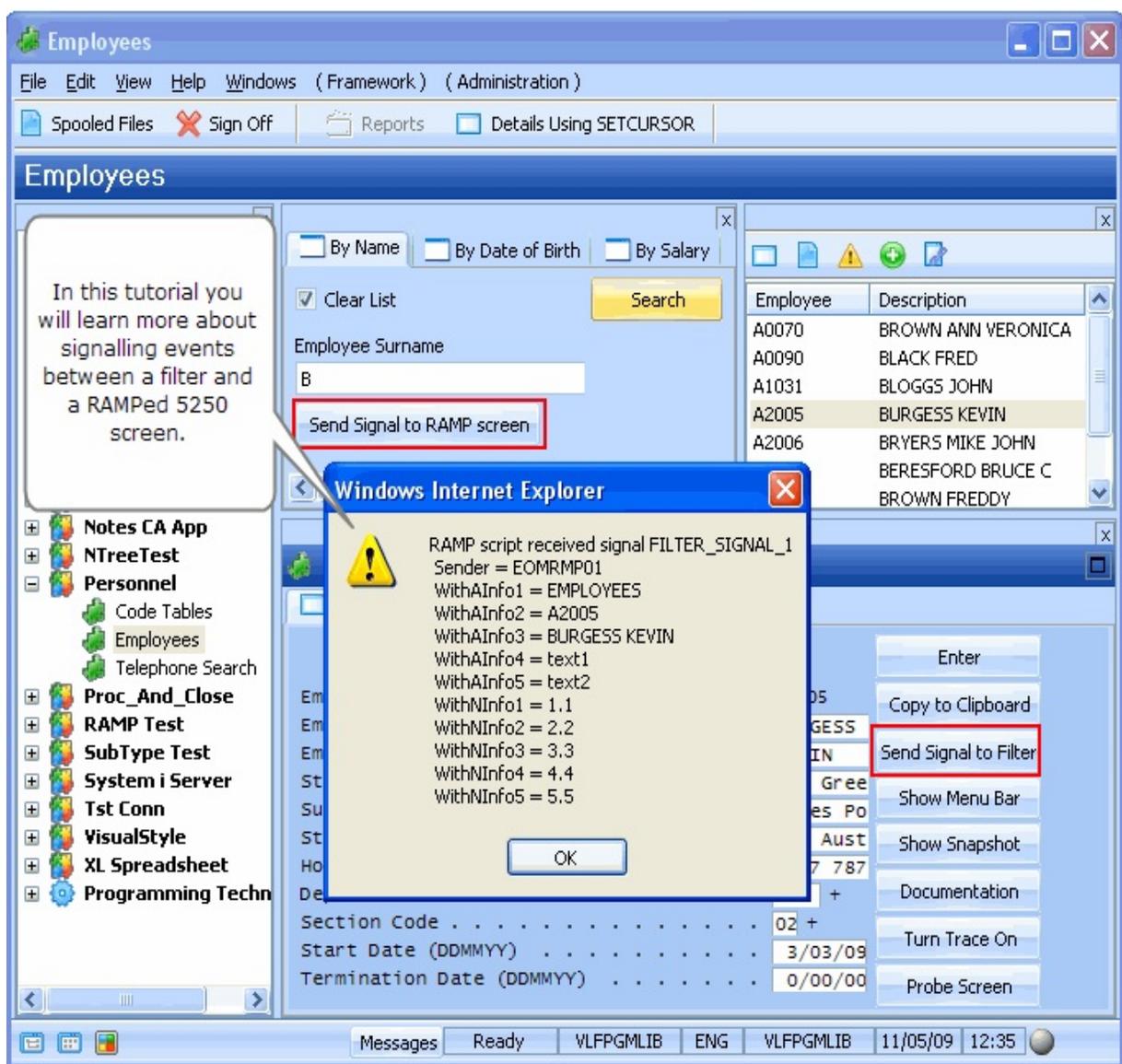
Objectives

Learn how RAMP screens signal events to the Framework

Learn how the Framework listens to RAMP signals

Learn how the Framework signals events to the RAMP screen

Learn how the RAMP screen listens to Framework signals



To achieve this objective, you will complete the following steps:

[RAMP-TSAD02 Step 1. Add a Signal Button to the By Name Filter](#)

[RAMP-TSAD02 Step 2. Make Your 5250 Screen Listen to the Signal](#)

[RAMP-TSAD02 Step 3. Test Signaling from Filter to RAMP Screen](#)

[RAMP-TSAD02 Step 4. Add a Signal Button to the RAMP Screen](#)

[RAMP-TSAD02 Step 5. Make the Filter Listen to the Signal](#)

[RAMP-TSAD02 Step 6. Signalling from a RAMP script to a VLF component](#)

[Summary](#)

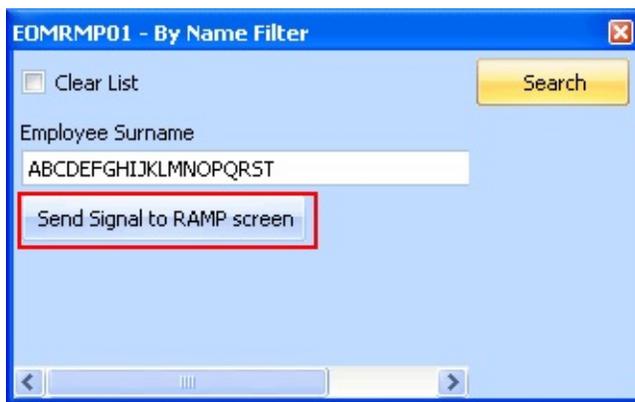
Before You Begin

In order to complete this tutorial, you must have completed the core tutorials RAMP-TS001 - RAMP-TS015.

RAMP-TSAD02 Step 1. Add a Signal Button to the By Name Filter

In this step you will modify the filter you created in [RAMP-TS003: Creating a Data Filter for Employees](#) so that it has an additional button which sends a signal to a RAMPed 5250 screen.

1. Open the filter reusable part iiiRMP01.
2. Drag a button from the Common Controls tab to the filter and make its caption Send Signal to RAMP Screen.



2. Add this code to the filter source to handle the click event of the button:

* Handle the signal 1 button by broadcasting FILTER_SIGNAL_1 with 5 alpha and 5 numeric payload items

```
EVTROUTINE HANDLING(#PHBN_1.Click)
```

```
Invoke #avListManager.GetCurrentInstance AKey1(#vf_elxak1)  
AKey2(#vf_elxak2) AKey3(#vf_elxak3) AKey4(#vf_elxak4)  
AKey5(#vf_elxak5) NKey1(#vf_elxnk1) NKey2(#vf_elxnk2)  
NKey3(#vf_elxnk3) NKey4(#vf_elxnk4) NKey5(#vf_elxnk4)  
Found(#vf_elBool) VisualId1(#VF_ELXVI1) VisualId2(#VF_ELXVI2)  
BusinessObjectType(#vf_elidn)
```

```
Invoke #Com_Owner.avSignalEvent withId(FILTER_SIGNAL_1)  
To(FRAMEWORK) SendAInfo1(#com_Owner.avObjectType)  
SendAInfo2(#vf_elxak1) SendAInfo3(#VF_ELXVI2) SendAInfo4("text1")  
SendAInfo5("text2") SendNInfo1(1.1) SendNInfo2(2.2) SendNInfo3(3.3)
```

SendNInfo4(4.4) SendNInfo5(5.5)

ENDROUTINE

Your code will look like this:

```
* Handle the signal 1 button by broadcasting FILTER_SIGNAL_1 with 5 alpha and 5 numeric payload items
↳EVTRoutine HANDLING(#PHEN_1.Click)

    Invoke #avListManager.GetCurrentInstance AKey1(#vf_elxak1) AKey2(#vf_elxak2) AKey3(#vf_elxak3)
        AKey4(#vf_elxak4) AKey5(#vf_elxak5) NKey1(#vf_elxnk1) NKey2(#vf_elxnk2) NKey3(#vf_elxnk3)
        NKey4(#vf_elxnk4) NKey5(#vf_elxnk4) Found(#vf_elBool) VisualId1(#VF_ELXVI1) VisualId2(#VF_ELXVI2)
        BusinessObjectType(#vf_elidn)

    Invoke #Com_Owner.avSignalEvent withId(FILTER_SIGNAL_1) To(FRAMEWORK)
        SendAInfo1(#com_Owner.avObjectType) SendAInfo2(#vf_elxak1) SendAInfo3(#VF_ELXVI2)
        SendAInfo4("text1") SendAInfo5("text2") SendNInfo1(1.1) SendNInfo2(2.2) SendNInfo3(3.3)
        SendNInfo4(4.4) SendNInfo5(5.5)

↳ENDROUTINE
```

↳End_Com

The GetCurrentInstance statement retrieves information of the currently selected entry in the instance list. The avSignalEvent then signals this information to the Framework.

Note that some of the values passed by the avSignalEvent are just static text or numbers. This is just to demonstrate that the command can pass five strings and five numeric values.

3. Compile the filter. (If the compile fails, it may be because the filter is being used in the Framework. If this is the case, restart the Framework).

RAMP-TSAD02 Step 2. Make Your 5250 Screen Listen to the Signal

In this step you will modify the script associated with the DisplayEmployee screen to listen for the FILTER_SIGNAL_1 and to display a message when the signal is received.

You will use the vHandle_AVEVENT function which listens for events coming from other components

1. If the Framework is not running, start it.
2. Start RAMP Tools and locate the script associated with the DisplayEmployee screen.
3. Add this code just above the SYSINFO block:

```
/* =====  
*/  
/* ===== AVEVENT =====  
*/  
/* =====  
*/  
vHandle_AVEVENT: function(WithId,Sender,WithAInfo1,WithAInfo2,WithA  
{  
var sText = "";  
  
if (WithId == "FILTER_SIGNAL_1")  
{  
sText += "RAMP script received signal " + WithId;  
sText += "\r Sender = " + Sender;  
sText += "\r WithAInfo1 = " + WithAInfo1;  
sText += "\r WithAInfo2 = " + WithAInfo2;  
sText += "\r WithAInfo3 = " + WithAInfo3;  
sText += "\r WithAInfo4 = " + WithAInfo4;  
sText += "\r WithAInfo5 = " + WithAInfo5;  
sText += "\r WithNInfo1 = " + WithNInfo1.toString();  
sText += "\r WithNInfo2 = " + WithNInfo2.toString();  
sText += "\r WithNInfo3 = " + WithNInfo3.toString();  
sText += "\r WithNInfo4 = " + WithNInfo4.toString();  
sText += "\r WithNInfo5 = " + WithNInfo5.toString();  
alert(sText);
```

```

}
return(true);
},

```

Your code should look like this:

```

/* Show details from clipboard */
ALERT_MESSAGE("Employee details from the clipboard are: ", "Employee Number: ", strEmpno,"Name: ", strGNar
),

```

```

/* ===== */
/* ===== AVEVENT ===== */
/* ===== */
vHandle_AVEVENT: function(withId,Sender,withAInfo1,withAInfo2,withAInfo3,withAInfo4,withAInfo5,withNInfo1,wi
{
var sText = "";

if (withId == "FILTER_SIGNAL_1")
{
sText += "RAMP script received signal " + withId;
sText += "\r Sender = " + Sender;
sText += "\r withAInfo1 = " + withAInfo1;
sText += "\r withAInfo2 = " + withAInfo2;
sText += "\r withAInfo3 = " + withAInfo3;
sText += "\r withAInfo4 = " + withAInfo4;
sText += "\r withAInfo5 = " + withAInfo5;
sText += "\r withNInfo1 = " + withNInfo1.toString();
sText += "\r withNInfo2 = " + withNInfo2.toString();
sText += "\r withNInfo3 = " + withNInfo3.toString();
sText += "\r withNInfo4 = " + withNInfo4.toString();
sText += "\r withNInfo5 = " + withNInfo5.toString();
alert(sText);
}
return(true);
},

```

```

//<SYSINFO>
vName : "DisplayEmployee",
vGUID : "63005AE2681E45B4A04C616B4DA0D5EE",

```

4. Commit the changes and do a partial save of the RAMP definition.

RAMP-TSAD02 Step 3. Test Signaling from Filter to RAMP Screen

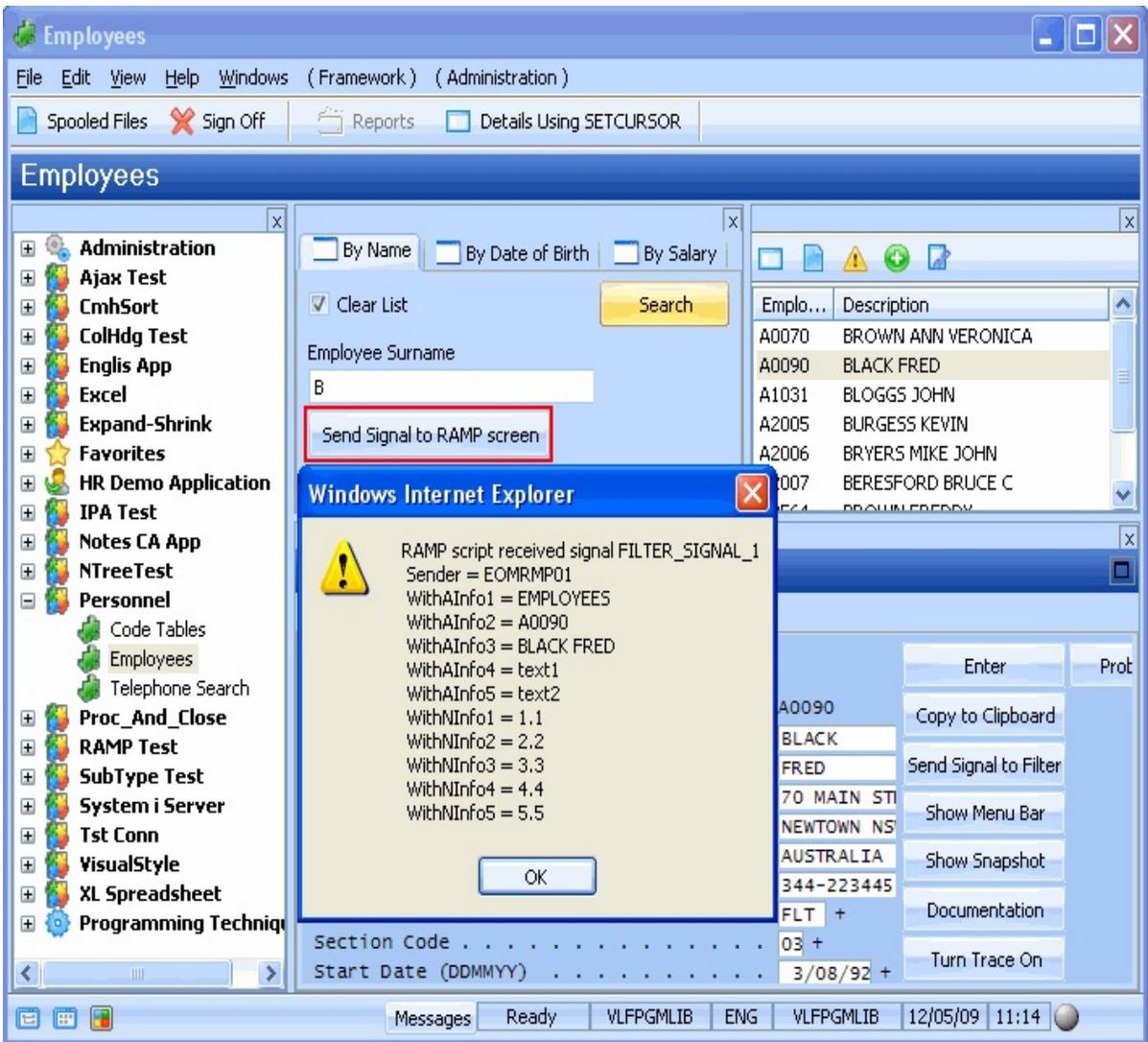
In this step you will test the button you have added to the filter.

1. Switch to the Framework window.
2. Select the Employees business object in the Personnel application.
3. In the By Name filter, click on the Send Signal to RAMP Screen button.

Notice that nothing happens. This is because there is no RAMP screen visible to receive the signal.

4. Use the filter to populate the instance list and select an employee in the list and wait for the Browse/Maintain Employee and Skill Files screen to appear.
5. Now click the Send Signal to RAMP Screen button again.

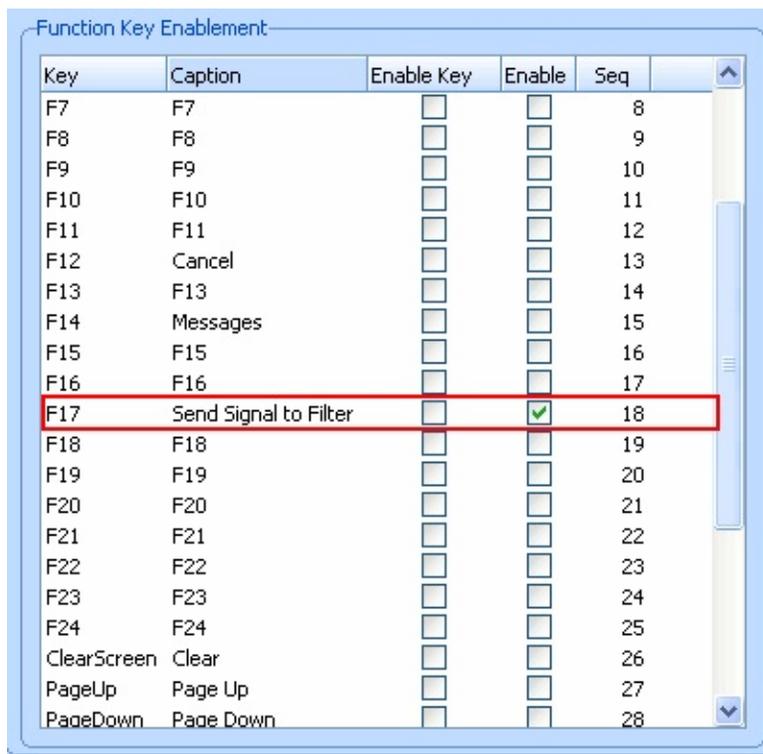
The filter fires off a Framework-wide signal. The DisplayEmployee screen's vHandle_AVEVENT function hears this signal and displays a message box indicating that it has received the signal and what the payload was.



RAMP-TSAD02 Step 4. Add a Signal Button to the RAMP Screen

In this step you will modify the DisplayEmployee screen so that it has a button that sends a signal to the Framework.

1. In the RAMP Tools window display the details of the DisplayEmployee screen.
2. In the Function Key Enablement section enable the F17 button and make its caption Send Signal to Filter.



Next you need to add the signal code to the button script:

3. In the vHandle_BUTTONCLICK function add this Case statement for the F17 key:

```
case KeyF17: /* Send RAMP_SIGNAL_1 */
    objGlobal.txtEmpno = GETVALUE("txtEmpno");
    objGlobal.txtGivenname = GETVALUE("txtGivenname");
    objGlobal.txtSurname = GETVALUE("txtSurname");
    AVSIGNALEVENT("RAMP_SIGNAL_1","FRAMEWORK",objG
break;
```

Your code will look like this:

```
case KeyF14:
    SENDKEY(KeyF14);
    break;
case KeyF17: /* Send RAMP_SIGNAL_1 */
    objGlobal.txtEmpno = GETVALUE("txtEmpno");
    objGlobal.txtGivename = GETVALUE("txtGivename");
    objGlobal.txtSurname = GETVALUE("txtSurname");
    AVSIGNALEVENT("RAMP_SIGNAL_1", "FRAMEWORK", objGlobal.txtEmpno, objGlobal.txtGivename, objGlobal.txtSurname);
    break;
default:
    SENDKEY(sbutton);
    break;
}
```

The code retrieves the value of the txtEmpno, txtGivename and txtSurname fields on the screen (you have named them in earlier tutorials) and sends a signal with this payload to the Framework.

It also sends some static text and numeric strings just to demonstrate that the AVSIGNALEVENT function can pass five strings and five numbers in its payload.

4. Commit the changes and do a partial save of the RAMP definition.
5. Save and restart the Framework.

RAMP-TSAD02 Step 5. Make the Filter Listen to the Signal

In this step you will add code to the filter to listen for the RAMP_SIGNAL_1 and to display a message showing the signal payload.

1. Open the iiiRMP01 reusable part in the Visual LANSAs editor.
2. Locate the EvtRoutine #Com_owner.avEvent event routine.
3. Add this code before the Endcase statement to listen for RAMP_SIGNAL_1 and issue a message with the signal payload:

```
when '= RAMP_SIGNAL_1'  
Use message_box_add ('VLF Filter received signal ' #EventId.Value)  
Use message_box_add (' WithAInfo1 =' #AInfo1)  
Use message_box_add (' WithAInfo2 =' #AInfo2)  
Use message_box_add (' WithAInfo3 =' #AInfo3)  
Use message_box_add (' WithAInfo4 =' #AInfo4)  
Use message_box_add (' WithAInfo5 =' #AInfo5)  
Use message_box_add (' WithNInfo1 =' #NInfo1)  
Use message_box_add (' WithNInfo2 =' #NInfo2)  
Use message_box_add (' WithNInfo3 =' #NInfo3)  
Use message_box_add (' WithNInfo4 =' #NInfo4)  
Use message_box_add (' WithNInfo5 =' #NInfo5)  
  
Use MESSAGE_BOX_SHOW
```

Your code will look like this:

```

-when '= Delete_List_Entry'
  Invoke Method(#avListManager.BeginListUpdate)

  * Remove instance details from the instance list
  |
  Invoke #avListManager.RemoveFromList AKey1(#EMPNO)
  Invoke Method(#avListManager.EndListUpdate)

-when '= RAMP_SIGNAL_1'
  Use message_box_add ('VLF Filter received signal ' #EventId.Value)
  Use message_box_add (' WithAInfo1 = ' #AInfo1)
  Use message_box_add (' WithAInfo2 = ' #AInfo2)
  Use message_box_add (' WithAInfo3 = ' #AInfo3)
  Use message_box_add (' WithAInfo4 = ' #AInfo4)
  Use message_box_add (' WithAInfo5 = ' #AInfo5)
  Use message_box_add (' WithNInfo1 = ' #NInfo1)
  Use message_box_add (' WithNInfo2 = ' #NInfo2)
  Use message_box_add (' WithNInfo3 = ' #NInfo3)
  Use message_box_add (' WithNInfo4 = ' #NInfo4)
  Use message_box_add (' WithNInfo5 = ' #NInfo5)

  Use MESSAGE_BOX_SHOW

-endcase
-Endroutine

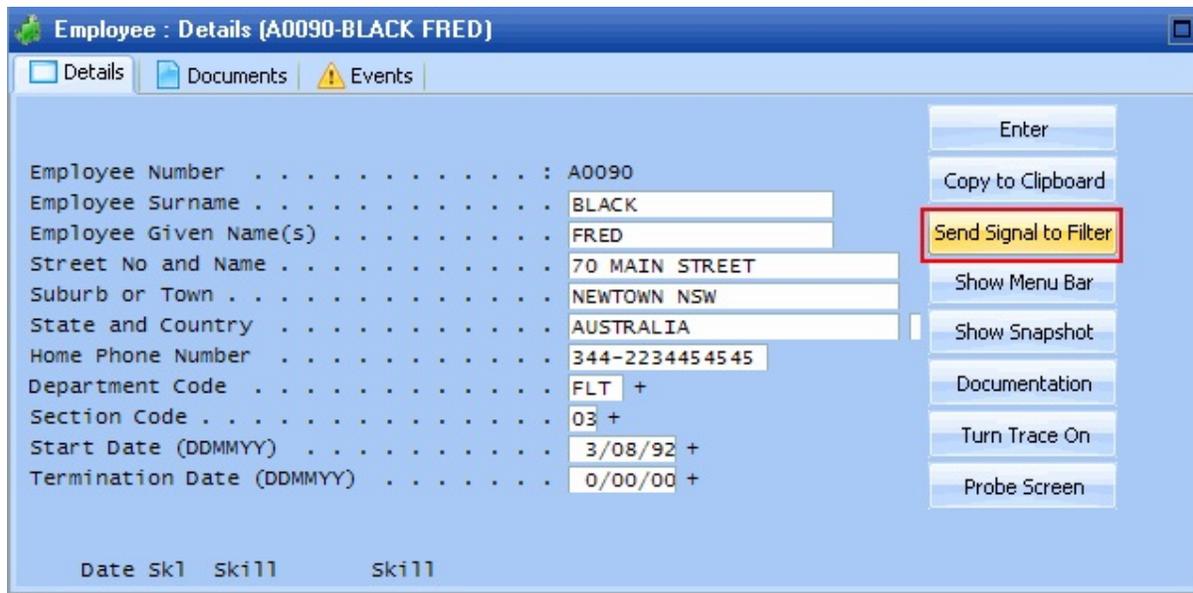
```

4. Compile the filter.

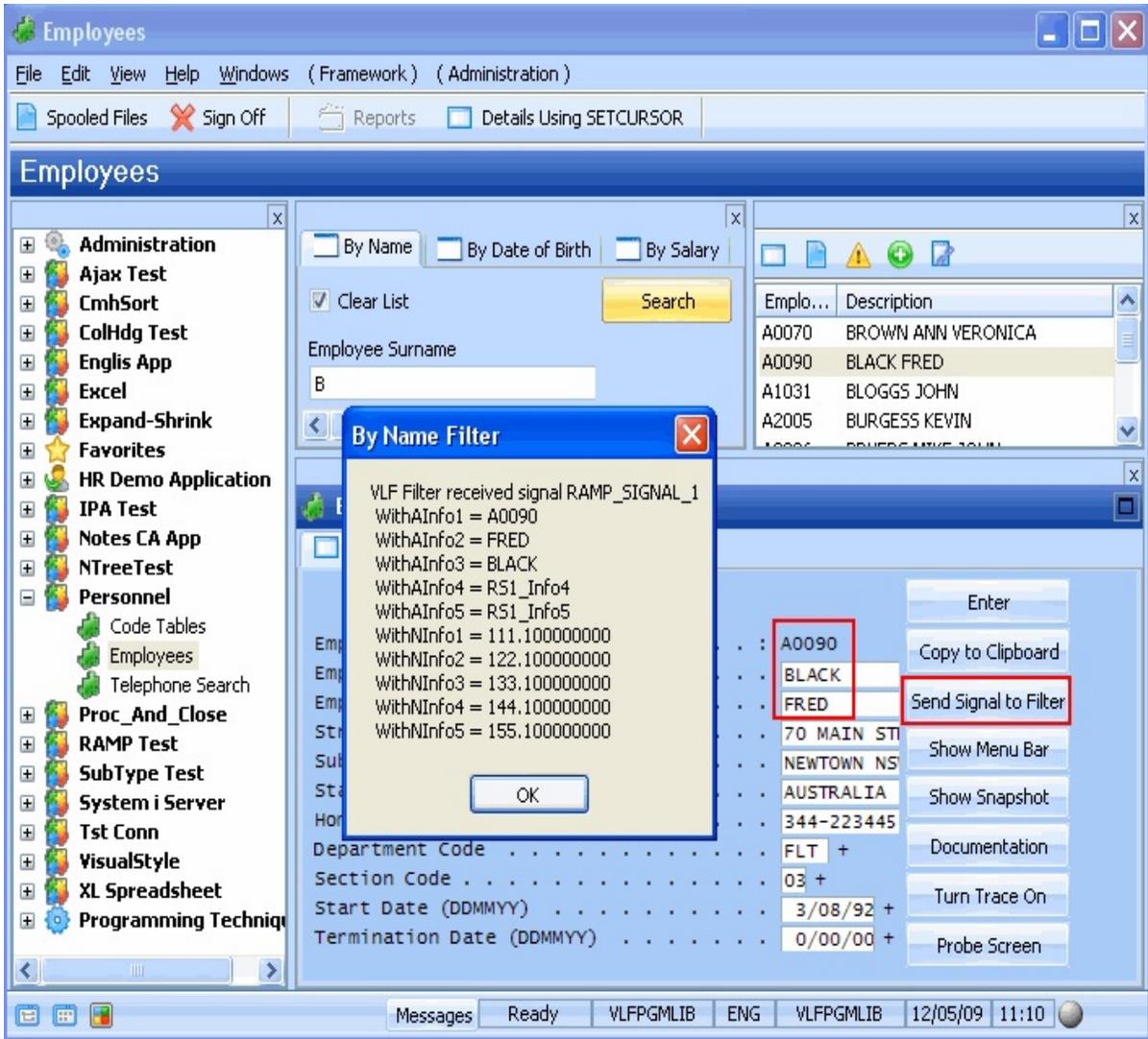
RAMP-TSAD02 Step 6. Signalling from a RAMP script to a VLF component

In this step you will test signaling from the RAMP script to the filter.

1. In the Framework select an employee from the instance list to redisplay the DisplayEmployee screen with the new button.
2. Click on the Send Signal to Filter button:



The RAMP script fires off a framework wide signal. The filter hears this signal in its EVTROUTINE HANDLING(#Com_Owner.avEvent) routine and displays message box indicating that it has received the signal, and what the payload was.



Summary

Important Observations

Signals are asynchronous. When you fire them and when they are received and actioned are not synchronous events. Sometimes they are synchronous, sometimes they are not. You should always code your applications as if they are asynchronous.

You should develop a naming standard for your signal identifiers and document all signals used and their associated payloads.

Signals are relatively expensive requests. They should action high level things like EMPLOYEE_UPDATED or LIST_DATA_CLEARED, rather than low level things like MOUSE_MOVED.

A RAMP destination screen's vHandle_AVEVENT function only listens for signals when it is the currently displayed 5250 screen (ie: only the current RAMP screen can listen for signals).

The second parameter (Sender) passed to vHandle_AVEVENT functions is only available in WIN applications. In WEB and .NET applications this parameter is not available and is always passed as an empty string.

Any RAMP destination screen may have its own unique vHandle_AVEVENT function.

What You Should Know

How to signal events between filters and RAMP command handlers

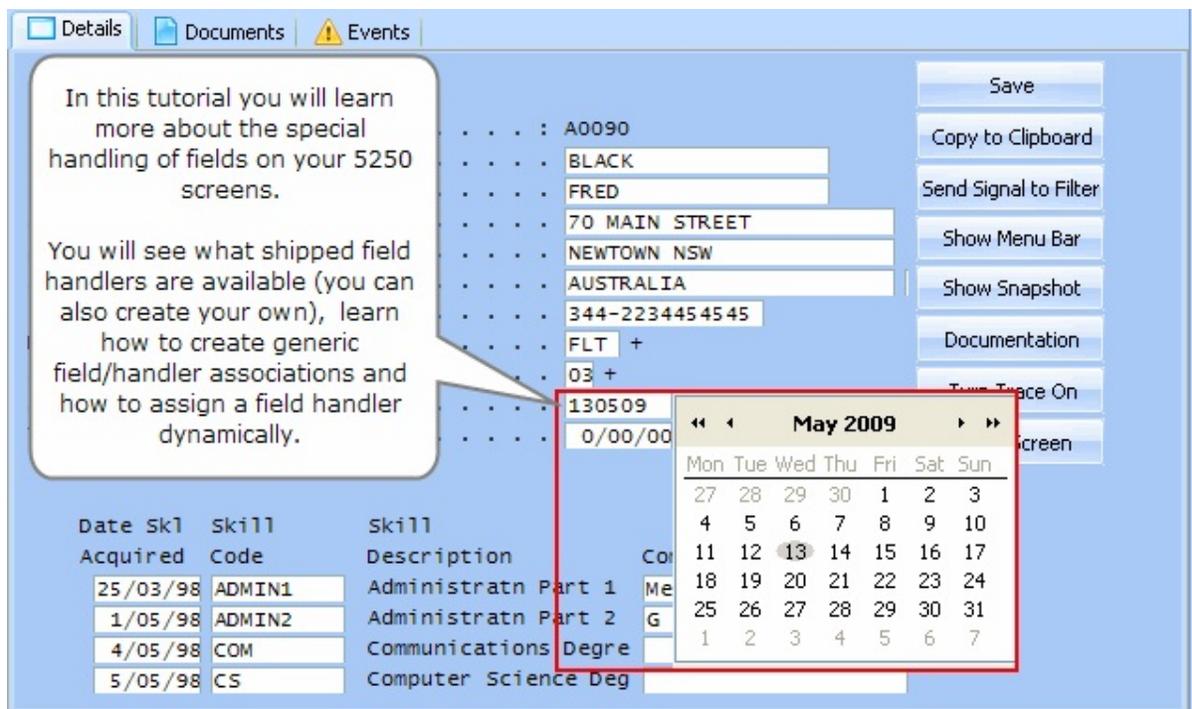
RAMP-TSAD03: Special Field Handling

RAMP's special field handling (prompting) allows you to add features and value to your existing 5250 screens. The special field handlers are Visual LANSA forms which execute on the PC allowing you to create functionality that would not be possible in a 5250 application.

Typically features can be added to many different 5250 screens by a just a single definition and naming standard.

Objectives

Demonstrate the features of automated prompting



To achieve this objective, you will complete the following steps:

[RAMP-TSAD03 Step 1. Understand What Makes the Prompter Appear](#)

[RAMP-TSAD03 Step 2. Being smarter with HANDLE_PROMPT\(\)](#)

[RAMP-TSAD03 Step 3. Handler Styles](#)

[RAMP-TSAD03 Step 4. Generic Handler Association](#)

[RAMP-TSAD03 Step 5. Generically Associating Date Fields with Date Picker](#)

[RAMP-TSAD03 Step 6. Dynamic Handler Association](#)

[RAMP-TSAD03 Step 7. Communicating with a Handler](#)

RAMP-TSAD03 Step 8. What to Do When Things Do Not Work

Summary

Before You Begin

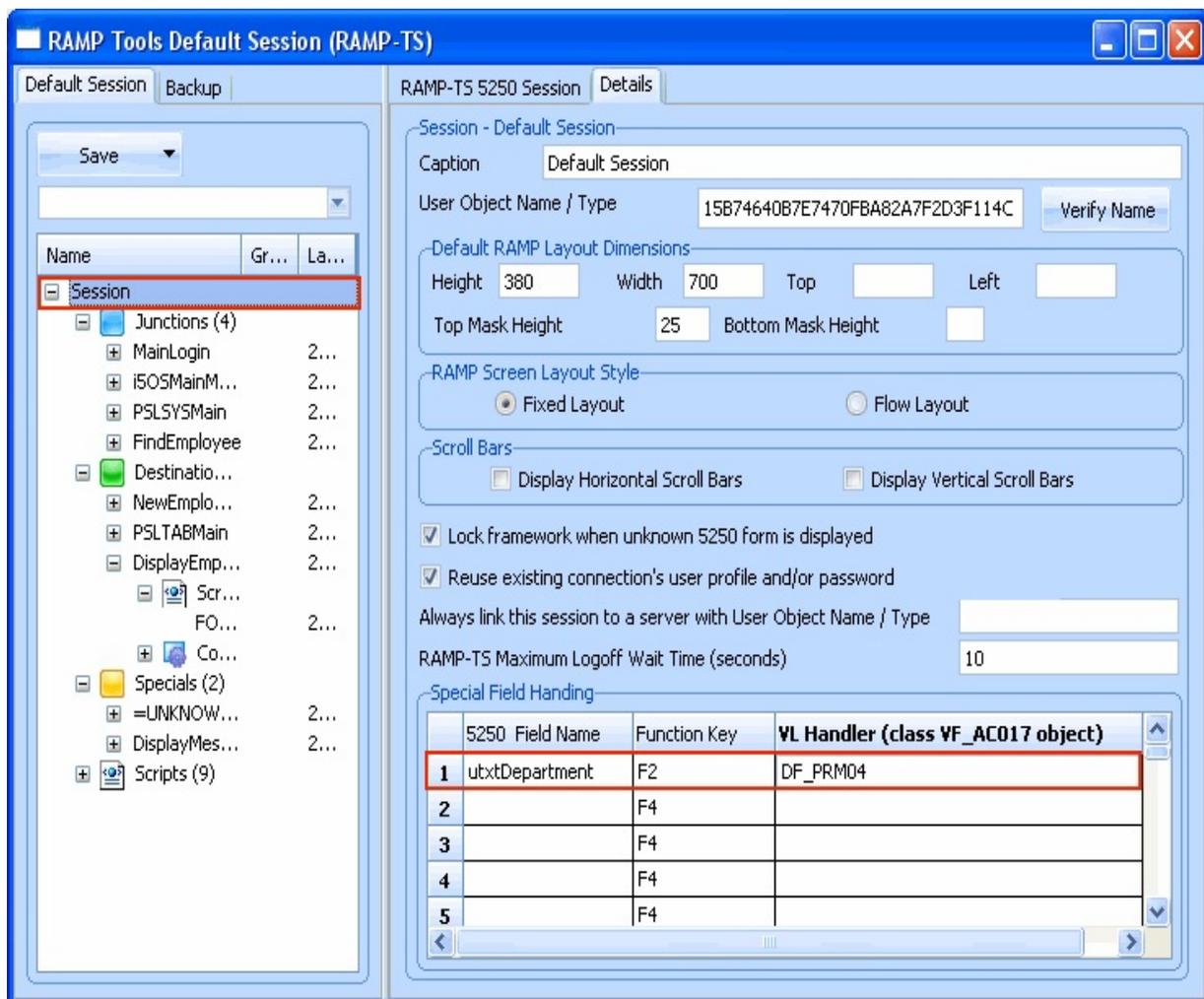
In order to complete this tutorial, you must have completed the core tutorials RAMP-TS001 - RAMP-TS015.

-

RAMP-TSAD03 Step 1. Understand What Makes the Prompter Appear

As you learnt in [RAMP-TS010: Using Special Field Handlers](#), automated prompting is set up in a table associated with a RAMP-TS session.

1. Open RAMP Tools and click on the Session node in the [Screen and Script List](#) on the left.
2. Locate the Special Field Handling table in the session details. You have defined in the table that if a field named utxtDepartment is on any 5250 screen, and the user presses F2, the special field handler DF_PRM04 is invoked.



DF_PRM04 is a Visual LANSA component. It is present, with source code, in

your VL environment.

In this case the function key you have chosen, F2, is not allowed by the 5250 screen. When the user presses the key, the request is intercepted and handled entirely on the client PC.

However, even if you use a key/button that is allowed by the 5250 screen, and you associate a special handler for the field with this key, the request will be intercepted and handled on the client PC. To understand why this is so:

3. Open the script associated with the DisplayEmployee screen.
4. Locate the vHandle_BUTTONCLICK function:

```
/* ===== */
/* =====  BUTTONCLICK  ===== */
/* ===== */
/* sButton: The button that was clicked */
vHandle_BUTTONCLICK: function(sButton)
{
    var bReturn = true;
    if (HANDLE_PROMPT()) return(bReturn); /* If the focus element is automatically prompted finish now */
    /* <BUTTONCLICK /> - Do not remove or alter this line */
}
```

The HANDLE_PROMPT() function call invokes a RAMP supplied function.

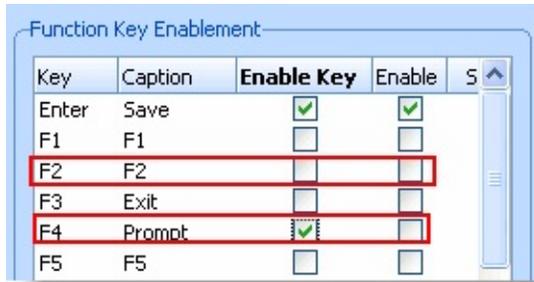
Using the name of the focus field on the 5250 screen and the function key used, it works out whether it should call a special field handler:

If it does, HANDLE_PROMPT() returns true – it handled the request – which is why the button script immediately terminates via return operation.

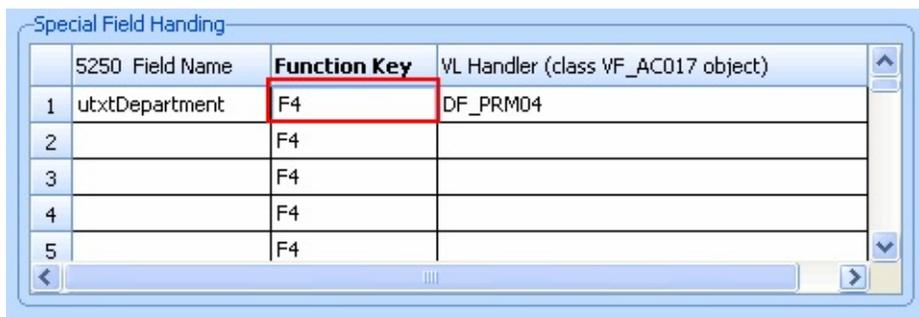
If HANDLE_PROMPT() did not invoke a special field handler it returns false, the button script continues to process the button click using its own logic.

In most cases the prompt request will be sent on to the 5250 screen for handling by the server because there will be no special field handler.

5. To test this out, disable the F2 key and enable the F4 key in the DisplayEmployee screen.



6. Save your changes and restart the Framework.
7. Locate the Employees business object in the Personnel application and display the details of an employee.
8. Press F4 on the Department Code field. Notice that the 5250 prompter is displayed.
9. Start the RAMP Tools and change the special field handling for the utxtDepartment field to use F4.



10. Save the RAMP definition and restart the Framework.
11. Display the details of an employee in the Personnel application and press F4. The special field handler is displayed.

Details Documents Events

Employee Number : A0090
 Employee Surname : BLACK
 Employee Given Name(s) : FRED
 Street No and Name : 70 MAIN STREET
 Suburb or Town : NEWTOWN NSW
 State and Country : AUSTRALIA
 Home Phone Number : 344-223445 4545
 Department Code : FLT + INTERNAL AUDITING
 Section Code : 01 + FLEET ADMINISTRATION
 Start Date (DDMMYY) : 3/0 ADMINISTRATION
 Termination Date (DDMMYY) : 0/0 PURCHASING
 ACCOUNTING

Date Acquired	Skill Code	Skill Description
25/03/98	ADMIN1	Administratn Part 1
1/05/98	ADMIN2	Administratn Part 2
4/05/98	COM	Communications Degre
5/05/98	CS	Computer Science Deg

RAMP-TSAD03 Step 2. Being smarter with HANDLE_PROMPT()

Note: This step simply demonstrates a technique. Please do not make this modification in your script.

In the preceding example HANDLE_PROMPT() was executed at the start of every button click.

However, if you know that all prompting on this 5250 screen will be done by using F4, you can make the logic faster and smarter like this:

```
vHandle_BUTTONCLICK: function(sButton)
{
    var bReturn = true;

    /* <BUTTONCLICK /> - Do not remove or alter this line */

    /* Handle function keys and buttons */

    switch (sButton)
    {
        case KeyF4:
            if (!HANDLE_PROMPT()) ALERT_MESSAGE("Position the cursor i
            break;

        case KeyEnter:
            SENDKEY(KeyEnter);

        etc, etc, etc
```

Here the HANDLE_PROMPT() request has been moved from the start of the button click function to the KeyF4 case statement so that it is only called when F4 is used, because we know that is the only time it is required.

Logic has also been added so that if HANDLE_PROMPT() does not handle the request (note the "!" in front of the function call) then a message box will appear saying "Position the cursor in a promptable field when using F4".

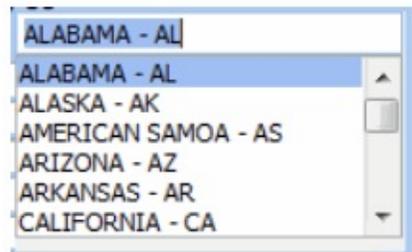
We can do this because we know that F4 is going to be handled exclusively on the client and not by the 5250 RPG program on the server.

This contrasts with default behavior in the preceding step that passed unhandled F4 requests on to the 5250 application to see if it wanted to handle them.

RAMP-TSAD03 Step 3. Handler Styles

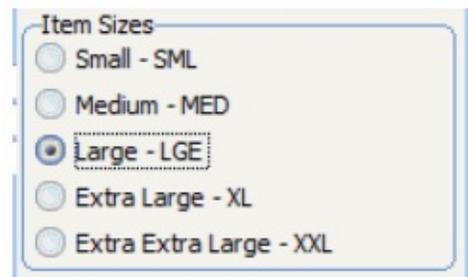
The way that a special field handler looks and acts is entirely up to you because you code them as Visual LANSA forms to do whatever you want.

Example handlers named DF_PRM01 – DF_PRM07 are shipped with the Framework. They demonstrate various handler behaviors that you might want to use as a basis for building your own handlers:



DF_PRM01

A simple list of Classic "code" s



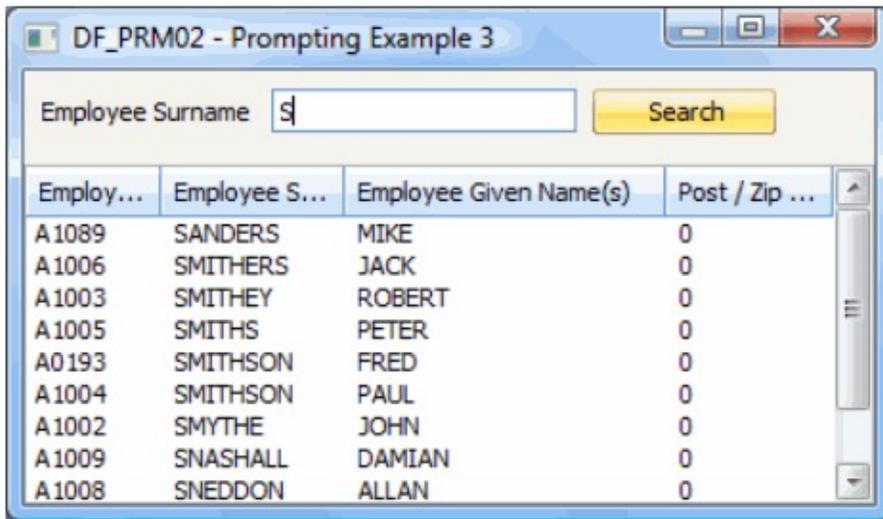
DF_PRM02

Selection of a c code from a list buttons.

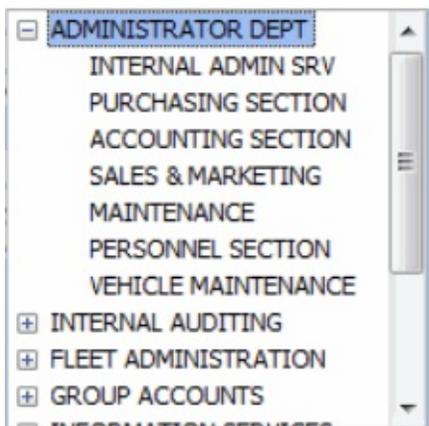
Classic "code" s

DF_PRM03

Locate an empl Classic "key" lo Product Numbe Number) used v data exists to us box or radio but

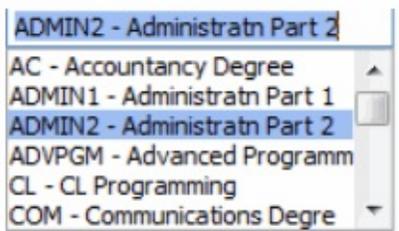


Often supports :
multiple ways (:
phone number, :
description, etc)



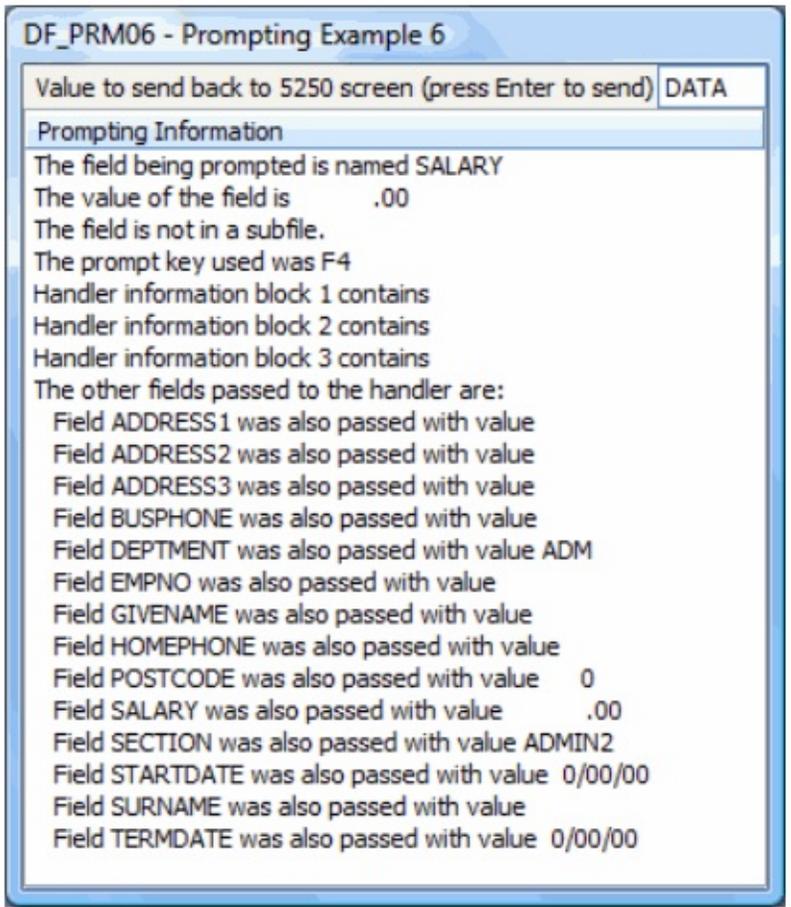
DF_PRM04

Selection of two
Slightly more a
"code" selector.



DF_PRM05

Like DF_PRM04
values are sent l
5250 form as th
selected. Shows
behavior of a ha
customized to v
prefer most.

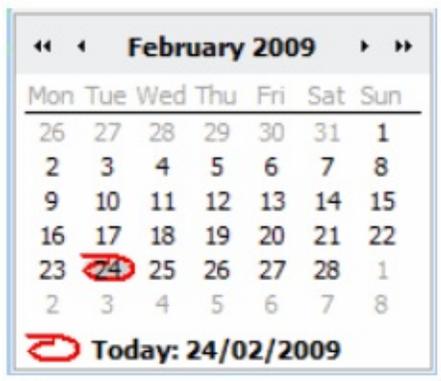


DF_PRM06

Generic debug
Displays details
information pas
handler.

Useful to run in
problematic spe
handler you are
check that the v
input to the han
you expect.

Use with applic
tracing.



DF_PRM07

Is a classic date
example is desi;
to 5250 fields n
DATE_nnnnnn_
nnnnn is a field
ffffff is the forr
should be proce

As shipped it sh
these date form.

CCYYsDDsMM

CCYYsMMsDI

CCYYDDMM

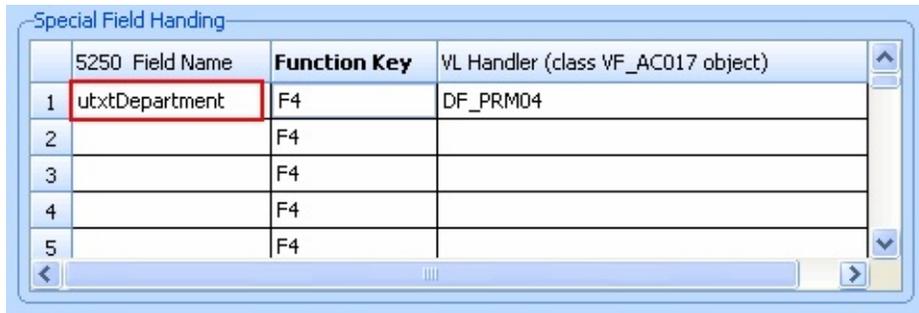
CCYYMM

CCYYMMDD
DDsMMsCCY'
DDsMMsYY
DDMMCCYY
DDMMYY
MMsDDsCCY'
MMsDDsYY
MMCCYY
MMDDCCYY
MMDDYY
MMYY
SysFmt6
SysFmt8
YYsMMsDD
YYMM
YYMMDD
Sample field na
work with are
DATE_Start_D
and
DATE_ORDER

Remember that you can apply whatever look and feel you like to your handlers. These are just shipped examples to help you get started. If you don't like the look or the behavior, change it.

RAMP-TSAD03 Step 4. Generic Handler Association

In the initial step of this tutorial you created a specific association between your 5250 field and the special field handler DF_PRM04 like this:



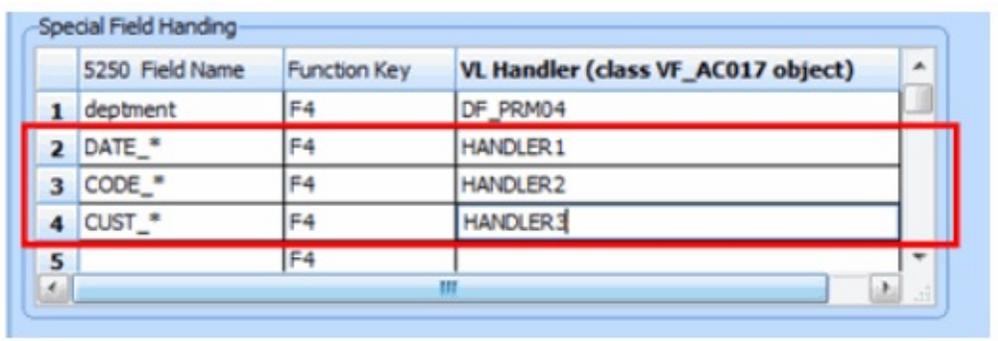
	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)
1	utxtDepartment	F4	DF_PRM04
2		F4	
3		F4	
4		F4	
5		F4	

This means that on any 5250 screen containing a field with this name the handler DF_PRM04 is called.

So by using a naming convention you can instantly add prompting to any 5250 screens that have a field with this name on them.

This type of specific association is most useful on key fields (like CustomerNumber, ProductNumber, etc) and code fields (like StateCode, CurrencyCode, etc). For key fields you can often associate a handler that allows the user to search in many different ways to locate a customer or product. For code fields you can often display the code/decode table allowing the user to select the code they want to use.

You can also create generic associations:



	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)
1	deptment	F4	DF_PRM04
2	DATE_*	F4	HANDLER.1
3	CODE_*	F4	HANDLER.2
4	CUST_*	F4	HANDLER.3
5		F4	

Here:

Fields starting with the name DATE_ will cause HANDLER1 to be invoked

when F4 is used.

Fields starting with the name `CODE_` will cause `HANDLER2` to be invoked when F4 is used.

Fields starting with the name `CUST_` will cause `HANDLER3` to be invoked when F4 is used.

If `HANDLER1` was a date picker, then you can see how by using a naming standard for your fields you can instantly associate `HANDLER1` with any date fields on your 5250 screen.

(Note that you can also create date handlers, drop-downs etc. using aXes eXtensions. For an introduction to aXes, see [RAMP-TSAD04: Redesigning the Screen Using aXes](#)).

Likewise, `HANDLER2` might substring off the rest of the prompted field name (eg: `CODE_STATE`, `CODE_CURRENCY` or `CODE_AIRPORT`) and be able to work out which code table (States, Currencies or Airports) it should display for selection. This type of "super-prompter" is commonly used because, while the data source changes (ie: which code table is displayed) the method of displaying and selecting the code is usually identical.

In the final example, `CUST_*` demonstrates a generic type of special field handler that does something special, presumably with a "CUST" (customer).

The things it does could range from name searching, to printing details, to pasting screen values to the clipboard, to mapping customer into a MS-Excel document, to preparing an e-mail to the customer. This is why special field handling is a lot more than simple F4 prompting.

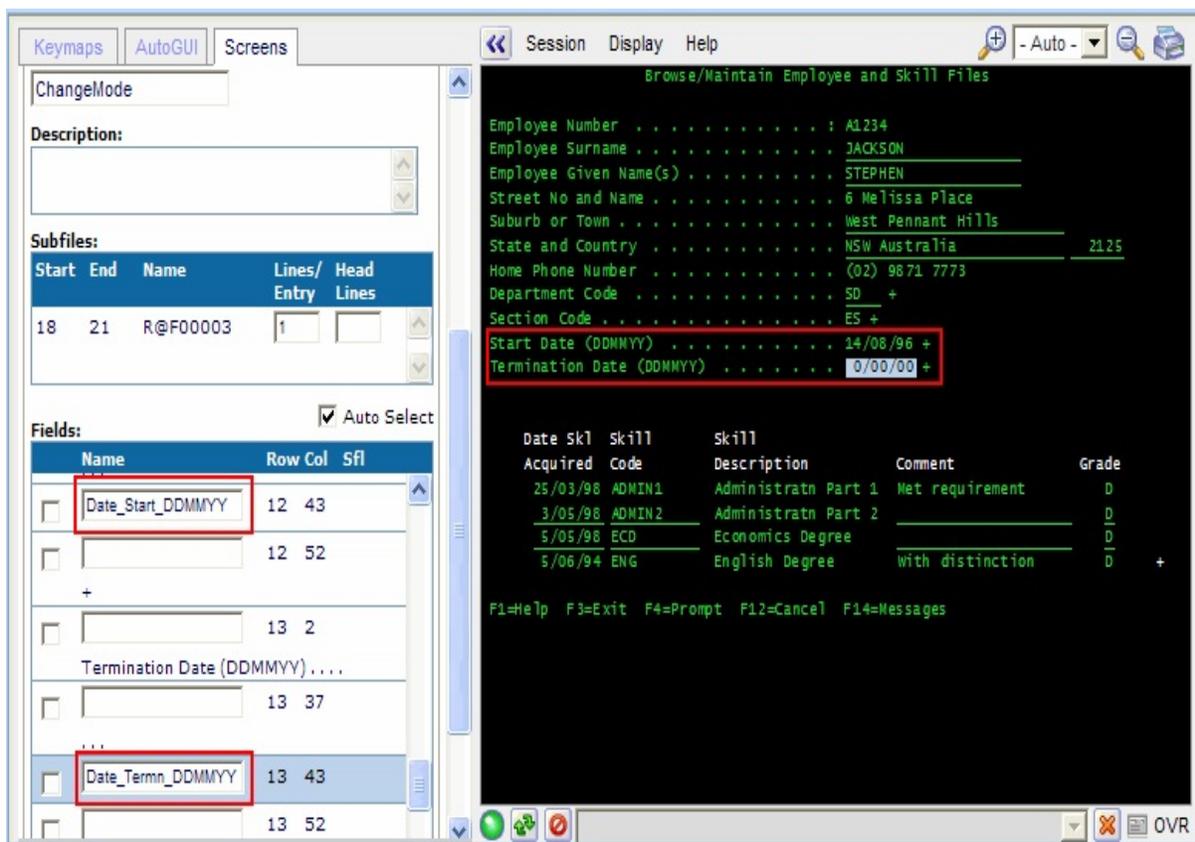
Special field handling is about attaching new behaviors to existing 5250 screens. By combining the special field handling table with a judicious naming standard you can attach new behaviors to many 5250 screens with no coding.

In these examples the naming standard chosen is used to communicate intention and information to the handler. There are also more precise ways of communicating intention and information to handlers. These are discussed in the later steps of this tutorial.

RAMP-TSAD03 Step 5. Generically Associating Date Fields with Date Picker

In this step you will generically associate date fields with the shipped date picker DF_PRM07. To test this, you will name the Start Date and Termination Date fields on the DisplayEmployee screen with a name starting with Date*.

1. Start the RAMP-TS 5250 emulator session in the RAMP Tools window.
2. Navigate to the Maintain/Browse Employee and Skill Files screen and press F21.
3. Name the Start Date field Date_Start_DDMMYY and save.
4. Name the Termination Date field Date_Termn_DDMMYY and save.



5. Display the Session details and associate fields named Date* with the DF_PRM07 date picker component.

Special Field Handling

	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)
1	utxtDepartment	F4	DF_PRM04
2	Date*	F4	DF_PRM07
3		F4	
4		F4	
5		F4	

6. Save the RAMP definition and restart the Framework.
7. Display the details of an employee and prompt for the Start Date and Termination Date fields. The special field handler DF_PRM07 is displayed for both fields (and for any other field named Date_nnnnnn_fffffff).

Details Documents Events

Employee Number : A2005

Employee Surname : BURGESS

Employee Given Name(s) : KEVIN

Street No and Name : 67 Green St

Suburb or Town : Blues Point

State and Country : NSW Australia

Home Phone Number : 9877 7877

Department Code : ADM +

Section Code : 02 +

Start Date (DDMMYY) : 130509

Termination Date (DDMMYY) : 0/00/00

Date sk1 skill skill
Acquired Code Description Col

0/00/00

0/00/00

0/00/00

0/00/00

Save

Copy to Clipboard

Send Signal to Filter

Show Menu Bar

Show Snapshot

Documentation

Turn Trace On

Screen

May 2009

Mon	Tue	Wed	Thu	Fri	Sat	Sun
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

RAMP-TSAD03 Step 6. Dynamic Handler Association

In the preceding steps you learnt how to permanently define a special field handler via RAMP Tools.

You can also dynamically define, modify and delete special handlers in your RAMP scripts. Typically this is done in your logon screen script so that it happens just once. However, this feature may be used in individual screen scripts for specialized purposes.

To dynamically define or redefine a special field handler use the `SET_SPECIAL_FIELD_HANDLER()` function. To dynamically remove a special field handler use `DROP_SPECIAL_FIELD_HANDLER()`;

Note that dynamically removing a handler will not impact it if is currently displayed. It will just prevent it from being displayed again.

1. Start RAMP Tools and locate the script for the DisplayEmployee screen.
2. In the `vHandle_ARRIVE` function for the screen dynamically attach a new handler to the `utxtDepartment` field like this:

```
SET_SPECIAL_FIELD_HANDLER("utxtDepartment",KeyEnter,"DF_PRM0
```

Your code will look like this:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;

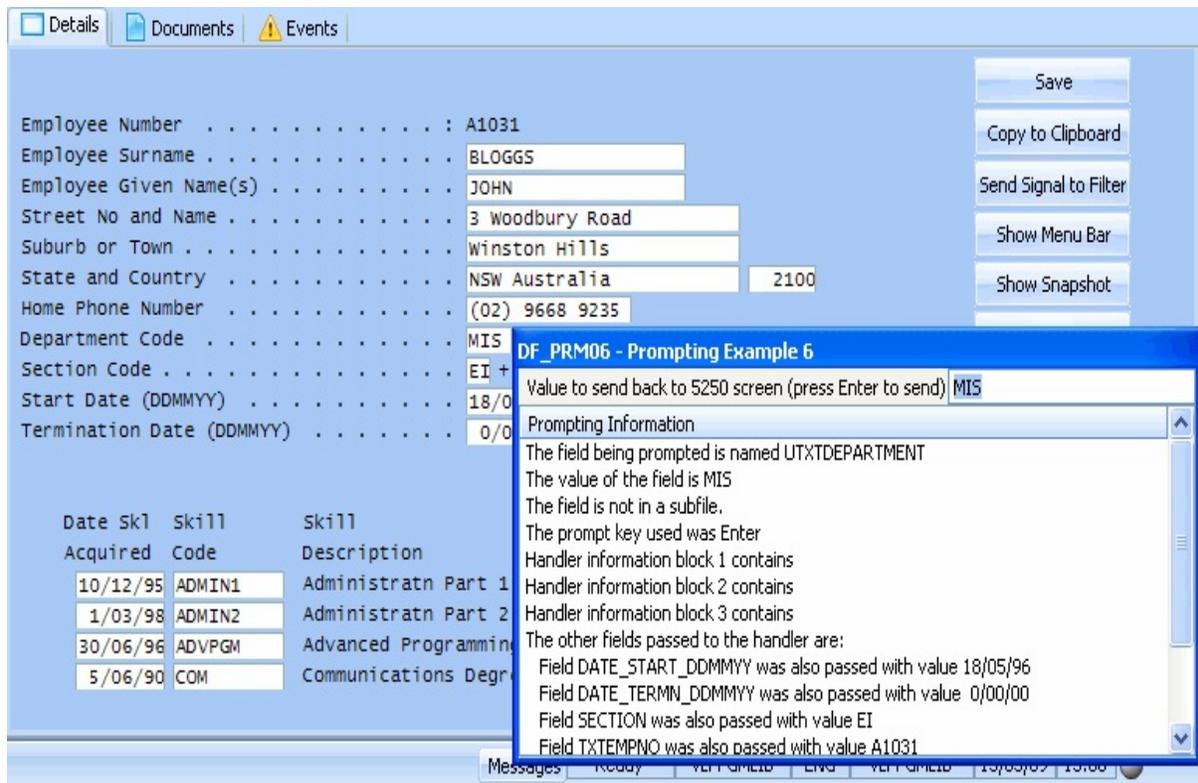
    if (CHECK_FIELD_EXISTS("txtSurname"))
    {
        SET_SPECIAL_FIELD_HANDLER("utxtDepartment",KeyEnter,"DF_PRM06"); /* Attach an Enter key handler */
        SHOW_CURRENT_FORM(true); /* Show the form in the framework and show VLF buttons */
        GET_FORM_MESSAGE(22); /* Extract messages and hide the message line */
        HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons displayed */
        SETBUSY(false); /* Last thing done - turn off the busy state */
    }
    else
    {
        SENDKEY(KeyF21);
    }

    /* <ARRIVE /> - Do not remove or alter this line */
}
```

The `DF_PRM06` handler will be invoked when the Enter key is pressed.

3. Commit the changes and do a partial save of the RAMP definition

4. Display the details of an employee in the Personnel application to show the screen with the modified script.
5. Press F4 on the Department Code field to cause the DF_PRM04 handler to be invoked from the field.
6. Press Enter on the Department Code field to cause the DF_PRM06 handler to be invoked from the field:



Note that this means you can attach multiple handlers to the same field, differentiated by the function key used.

7. Next drop the F4 handler associated with the field by adding this code to the vHandle_ARRIVE function:

```
DROP_SPECIAL_FIELD_HANDLER("utxtDepartment",KeyF4);
```

Your code will look like this:

```

var preturn = true;
if (CHECK_FIELD_EXISTS("txtsurname"))
{
SET_SPECIAL_FIELD_HANDLER("utxtDepartment",KeyEnter,"DF_PRM06"); /* Attach an Enter Key handler */
DROP_SPECIAL_FIELD_HANDLER("utxtDepartment",KeyF4); /* Drop the F4 handler */
SHOW_CURRENT_FORM(true); /* Show the form in the framework and show VLF buttons */
GET_FORM_MESSAGE(22); /* Extract messages and hide the message line */
HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons displayed */
SETBUSY(false); /* Last thing done - turn off the busy state */
}
else
{
SENDKEY(KeyF21);
}

```

8. Try this by selecting another employee from the list, and confirm that F4 on the Department Code field no longer brings up the DF_PRM04 special command handler (it shows the 5250 prompter instead).
9. Confirm that you can use Enter to activate the DF_PRM06 handler.

Remember that it is unusual (ie: specialized) to do this in a destination arrival script. Most dynamic attachment is done just once, in the logon script, and it persists for the entire session.

RAMP-TSAD03 Step 7. Communicating with a Handler

There are two ways to communicate additional intent and information to a handler.

The first involves passing information on the HANDLE_PROMPT() request.

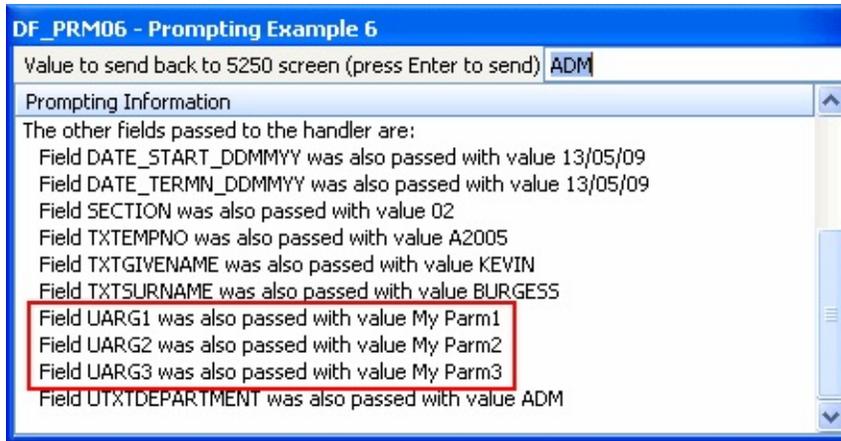
1. Start RAMP Tools.
2. In the script for the DisplayEmployee screen locate the button script's HANDLE_PROMPT() function call and add string parameters to the call like this:

```
HANDLE_PROMPT("My Parm1","My Parm2","My Parm3")
```

Your code will look like this:

```
vHandle_BUTTONCLICK: function(sButton)
{
    var bReturn = true;
    if (HANDLE_PROMPT("My Parm1", "My Parm2", "My Parm3")) return(bReturn); /* If tl
    /* <BUTTONCLICK /> - do not remove or alter this line */
```

3. Commit the changes and do a partial save of the RAMP definition.
4. In the Framework window, display the details of an employee to run the modified script for the screen.
5. Prompt the Department Code field by pressing Enter to display the testing/debugging handler DF_PRM06. The result you see looks like this:



The arguments passed to the `HANDLE_PROMPT()` function are passed on to the handler with the symbolic names `UARG1`, `UARG2`, `UARG3`, etc. The Visual LANSA handler can retrieve these values by using method calls like this in its code:

```
Invoke Method(#Com_Owner.uGet5250Field) Name(UARG1) Value(#Std_Te
```

This technique is fine when you know what handler you are talking to, or are talking to all handlers generically. This technique also means you probably need a convention for what `UARG1`, `UARG2`, etc are used for.

To communicate with a specific handler, use the next technique.

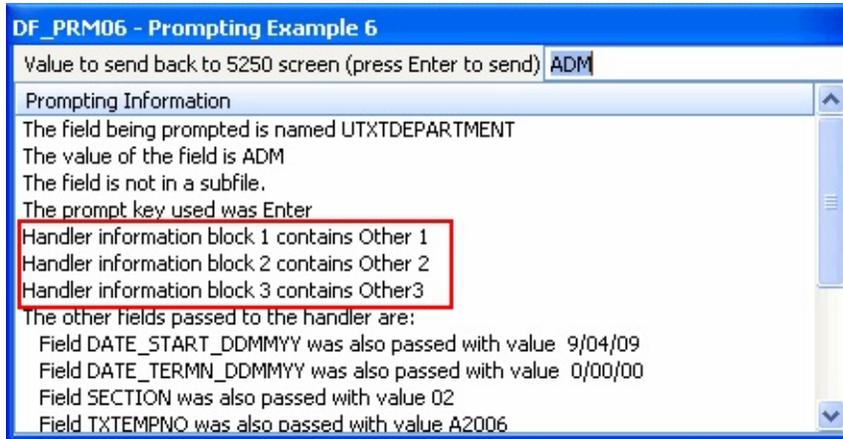
The second communication involves attaching information to the handler via the `SET_SPECIAL_FIELD_HANDLER()` function.

This is done by using the three optional parameters at the end of the function call.

6. Change the `SET_SPECIAL_FIELD_HANDLER` statement you created in the previous step to:

```
SET_SPECIAL_FIELD_HANDLER("utxtDepartment",KeyEnter,"DF_1  
"Other 1","Other 2","Other3");
```

7. Commit your changes and do a partial save, then select an employee to run the script for the screen again.
8. Cause the VF_PRM06 handler to be invoked. You will see this:



These three information blocks allow you to communicate with a precise handler. The Visual LANSA handler can retrieve these values as properties, like this example:

```
#Product := #Com_Owner.uHandlerInfo1  
#Customer := #Com_Owner.uHandlerInfo2  
#ZipCode = #Com_Owner.uHandlerInfo1.toNumber()
```

RAMP-TSAD03 Step 8. What to Do When Things Do Not Work

If you have a problem with special field handlers you will need to debug them.

Trying to debug them with `ALERT_MESSAGE()`, `alert()` or `MESSAGE_BOX_SHOW` operations will probably just confuse you.

There are two reasons for this.

First, `ALERT_MESSAGE` is asynchronous, so when it displays may not be when you think it does.

Secondly, using these operations presents a windows message box. These usually take focus. Most handlers hide themselves when they lose focus or become deactivated, so typically every time you show a message box your handler will disappear. The same may be true of using the VL code debugger. Debugging focus and activation sensitive applications can be tricky.

The best solution is to use the framework application level trace facility, positioned so that it does not overlap you framework window. If you just turn it on a great deal of special field handling trace information will appear.

If this does not resolve the problem then start adding `TRACE()` calls to your RAMP scripts and `#AvFrameworkManager.AvRecordTrace` in your VL code.

Another useful debugging feature is to plug in the shipped `DF_PRM06` handler in place of your handler. When invoked it reports on a lot of information about what information was passed to the handler, which may help you to identify your problem.

Summary

Important Observations

Special field handlers can do F4 type prompting very easily. The results usually look better, work faster, and place less load on your 5250 server.

Special field handlers can do a lot more than F4 prompting because they allow you to attach behaviors to fields on your 5250 forms in many ways and at many levels.

Application tracing is the tool you should use to debug handler issues

If you write down a list of all the "key" and "code" fields in your 5250 application and use a judicious field naming standard, you can automatically attach a prompting capability to all your 5250 screens.

Also see [Advanced Prompting](#).

What You Should Know

How to use special field handling to add value to your 5250 application

RAMP-TSAD04: Redesigning the Screen Using aXes

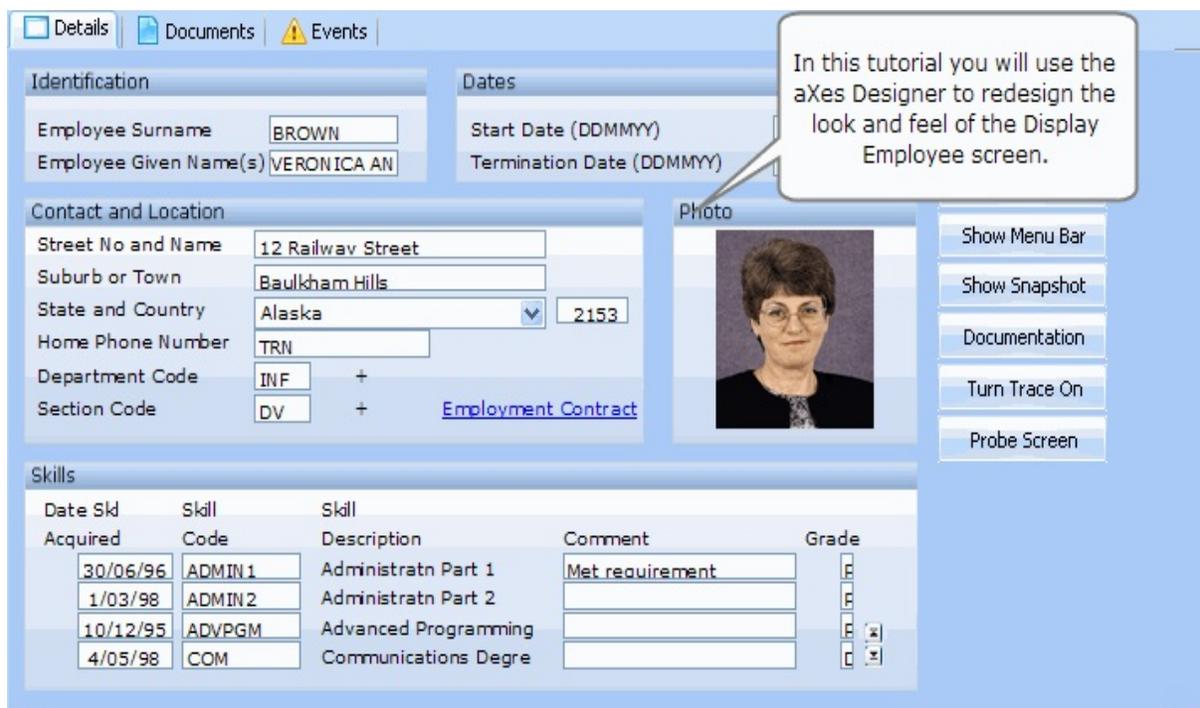
You can use the aXes Designer to freely redesign your 5250 screens. You can move and hide content, change labels, apply styles and use visual elements such as lines and group boxes.

Using aXes eXtensions, you can also display your content as drop-downs, calendar drop-downs, check boxes or radio buttons and add new elements such as buttons, images, tooltips, hyperlinks and Google maps.

You will notice that what you can do with aXes eXtensions partly overlaps with what you can do with, for example, special field handlers. It is up to you which of these approaches you adopt to enhance your screens.

Objectives

Learn how to use aXes Designer to redesign your screen.



The screenshot displays a 5250 screen with a light blue background. At the top, there are three tabs: 'Details' (selected), 'Documents', and 'Events'. The main content is organized into several sections:

- Identification:** Employee Surname: BROWN; Employee Given Name(s): VERONICA AN.
- Dates:** Start Date (DDMMYY); Termination Date (DDMMYY).
- Contact and Location:** Street No and Name: 12 Railway Street; Suburb or Town: Baulkham Hills; State and Country: Alaska (dropdown); Home Phone Number: TRN; Department Code: INF; Section Code: DV. A link for 'Employment Contract' is visible.
- Photo:** A portrait of a woman with short brown hair and glasses.
- Skills:** A table with columns: Date Skd, Skill Code, Skill Description, Comment, and Grade.

A tooltip box is positioned over the Photo section, containing the text: "In this tutorial you will use the aXes Designer to redesign the look and feel of the Display Employee screen." To the right of the photo, there are five buttons: 'Show Menu Bar', 'Show Snapshot', 'Documentation', 'Turn Trace On', and 'Probe Screen'.

Date Skd	Skill Code	Skill Description	Comment	Grade
30/06/96	ADMIN1	Administratn Part 1	Met requirement	P
1/03/98	ADMIN2	Administratn Part 2		P
10/12/95	ADVPGM	Advanced Programming		P
4/05/98	COM	Communications Degre		C

To achieve this objective, you will complete the following steps:

- [RAMP-TSAD04 Step 1. Get Started with aXes Designer](#)
- [RAMP-TSAD04 Step 2. Set up Styles](#)
- [RAMP-TSAD04 Step 3. Hide Repetitive Information](#)
- [RAMP-TSAD04 Step 4. Add a Tooltip](#)

- [RAMP-TSAD04 Step 5. Add a Drop-Down](#)
- [RAMP-TSAD04 Step 6. Organize Fields inside Group Boxes](#)
- [RAMP-TSAD04 Step 7. Add Up and Down Buttons to Subfile](#)
- [RAMP-TSAD04 Step 8. Hide Function Keys and Add a Picture](#)
- [RAMP-TSAD04 Step 9. Add a Hyperlink](#)
- [RAMP-TSAD04 Step 10. Test the Redesigned Screen](#)
- [RAMP-TSAD04 Step 11. Remove the Screen Customization](#)

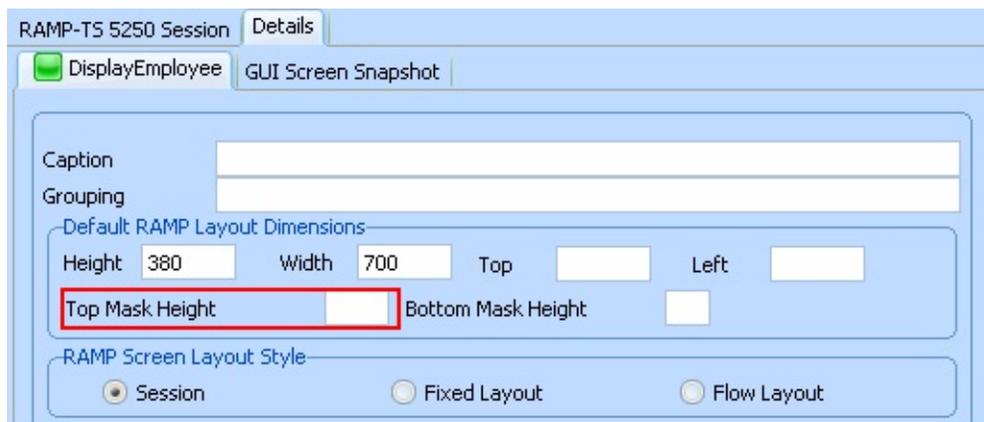
Before You Begin

In order to complete this tutorial, you must have completed the core tutorials RAMP-TS001 - RAMP-TS015.

RAMP-TSAD04 Step 1. Get Started with aXes Designer

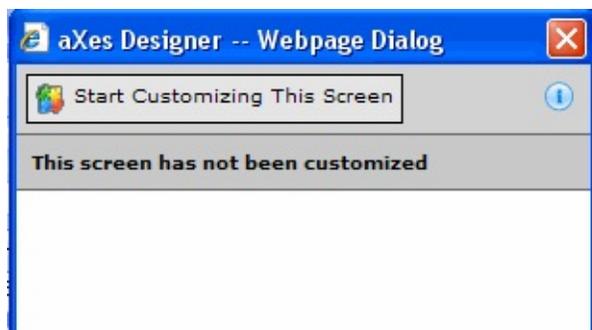
In this step you will learn how to start redesigning a screen using the aXes Designer window. You will also learn how to set the properties of elements on the screen.

1. In RAMP Tools, display the details of the DisplayEmployee screen.
2. In the Default RAMP Layout Dimensions section, ensure that the Top Mask Height is 0 or blank.



The top mask is used to hide the screen title in RAMP. In this tutorial you want the title displayed because you will learn how to hide it using aXes.

3. Start a RAMP-TS 5250 emulator session.
4. Navigate to the DisplayEmployee screen and display it in change mode.
5. In the aXes Designer window, click on Start Customizing This Screen.



The screen is now displayed without the 5250 emulator style. Notice that the

boundaries of the 5250 screen are indicated by a thin red line.

Browse/Maintain Employee and Skill Files

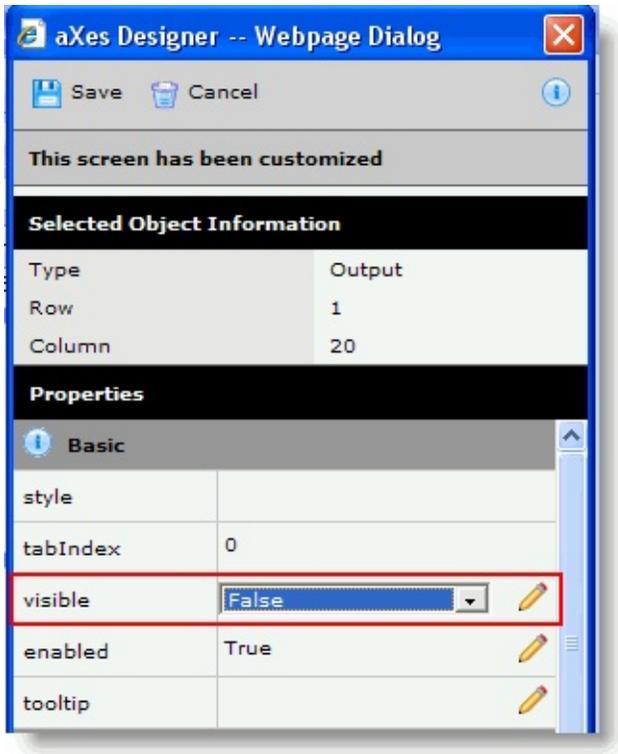
Employee Number : A1234
Employee Surname JACKSON
Employee Given Name(s) STEPHEN
Street No and Name 8 Melissa Place
Suburb or Town West Pennant Hills
State and Country NSW Australia 2125
Home Phone Number (02) 9871 7773
Department Code SD +
Section Code E9 +
Start Date (DDMMYY) 14/08/98 +
Termination Date (DDMMYY) 000/00 +

Date Skl	Skill	Skill			
Acquired	Code	Description	Comment	Grade	
25/03/98	ADMIN1	Administratn Part 1	Met requirement		
3/05/98	ADMIN2	Administratn Part 2			
5/05/98	ECD	Economics Degree			
5/08/94	ENG	English Degree	With distinction		

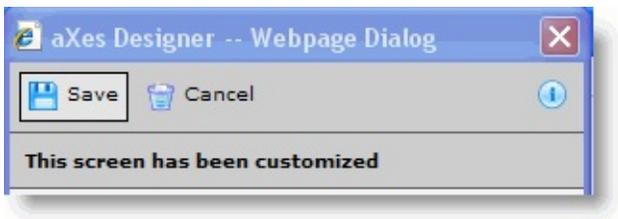
F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

You can change the boundaries of the screen by moving the red line. Refer to the aXes tutorials to learn how.

6. Click on the screen title to select it. Notice that the aXes Designer window now shows the properties of the screen title.
7. Set the Visible property of the title to False:



8. Click on the Save button on top of the aXes Designer window:



A message indicating that the screen customization has been saved is shown:



Notice the title is no longer displayed:

<< Session Display Help

Employee Number : A1234
Employee Surname : JACKSON
Employee Given Name(s) : STEPHEN

RAMP-TSAD04 Step 2. Set up Styles

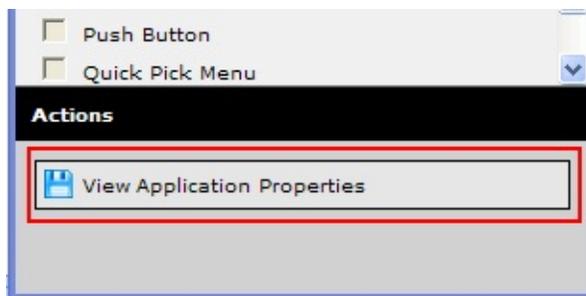
In this step you will set up a font style for all your screens.

You need to use styles in your application (as opposed to formatting individual screens or elements manually) because styles produce a common look and feel and prevent unconstrained style evolution. They also provide a single point of change so you can, for example, change the font of all screens by changing the style.

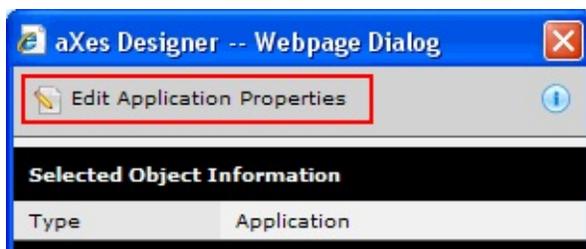
Even though you can freely customize screens using aXes, you should bear in mind that the look and feel of your application should be primarily controlled by the standard Framework themes, and that any customization needs to fit in with them.

For example, you should not use aXes styles to set up screen background colors, because the Framework automatically sets the background color of all RAMP screens.

1. Click on View Application Properties on the bottom of the aXes Designer window to display the properties of the application:



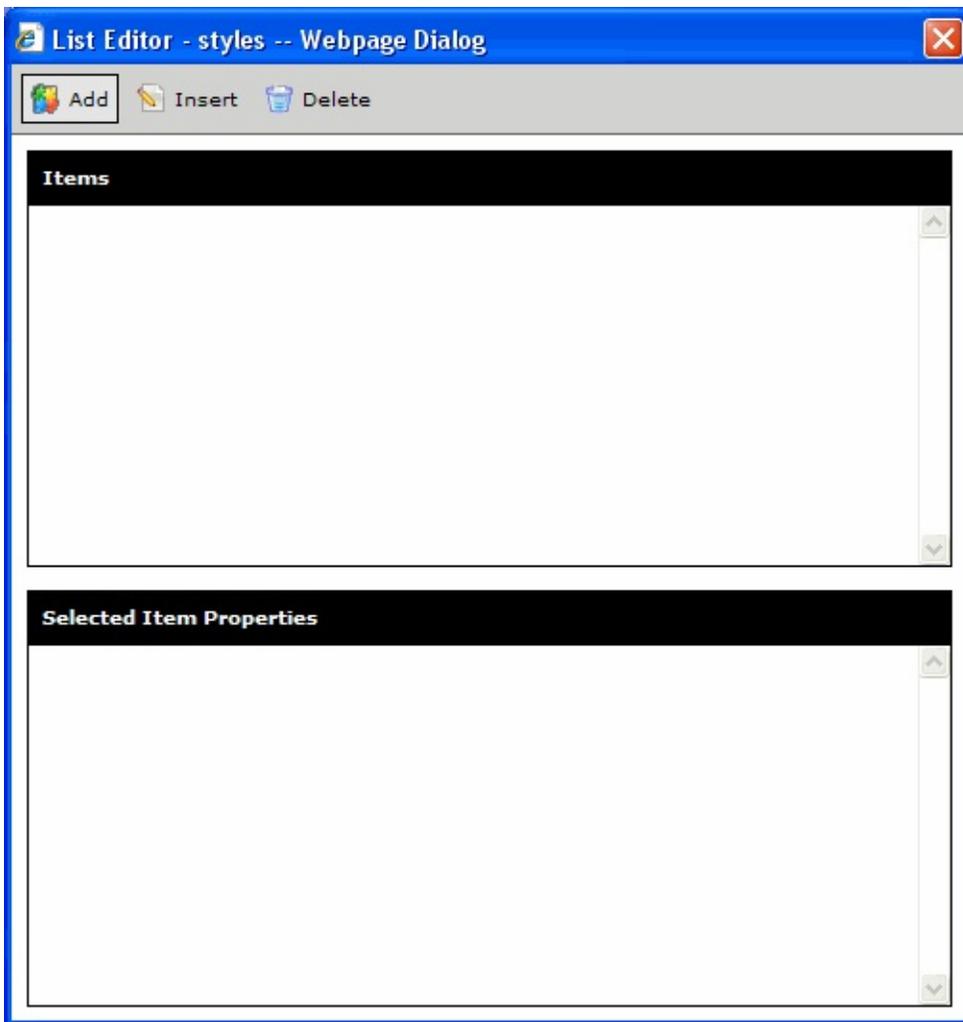
2. To edit the properties, click on Edit Application Properties on the top of the aXes Designer window:



3. Click on Styles in the Styling group.



The Styles editor is displayed:



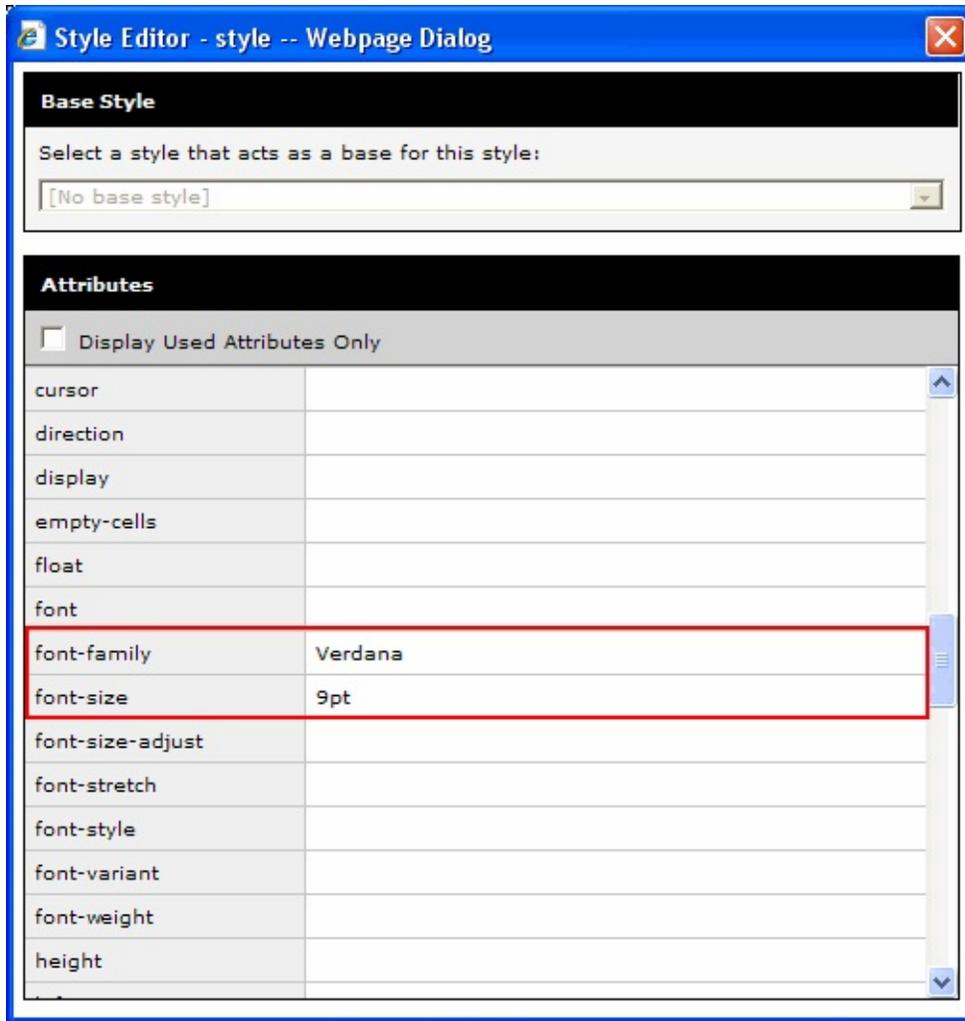
4. Click on the Add button to add a style:

Specify BasicFont as the name of the style
Select All as the value for StyleFor property
Leave the htmlTag property blank

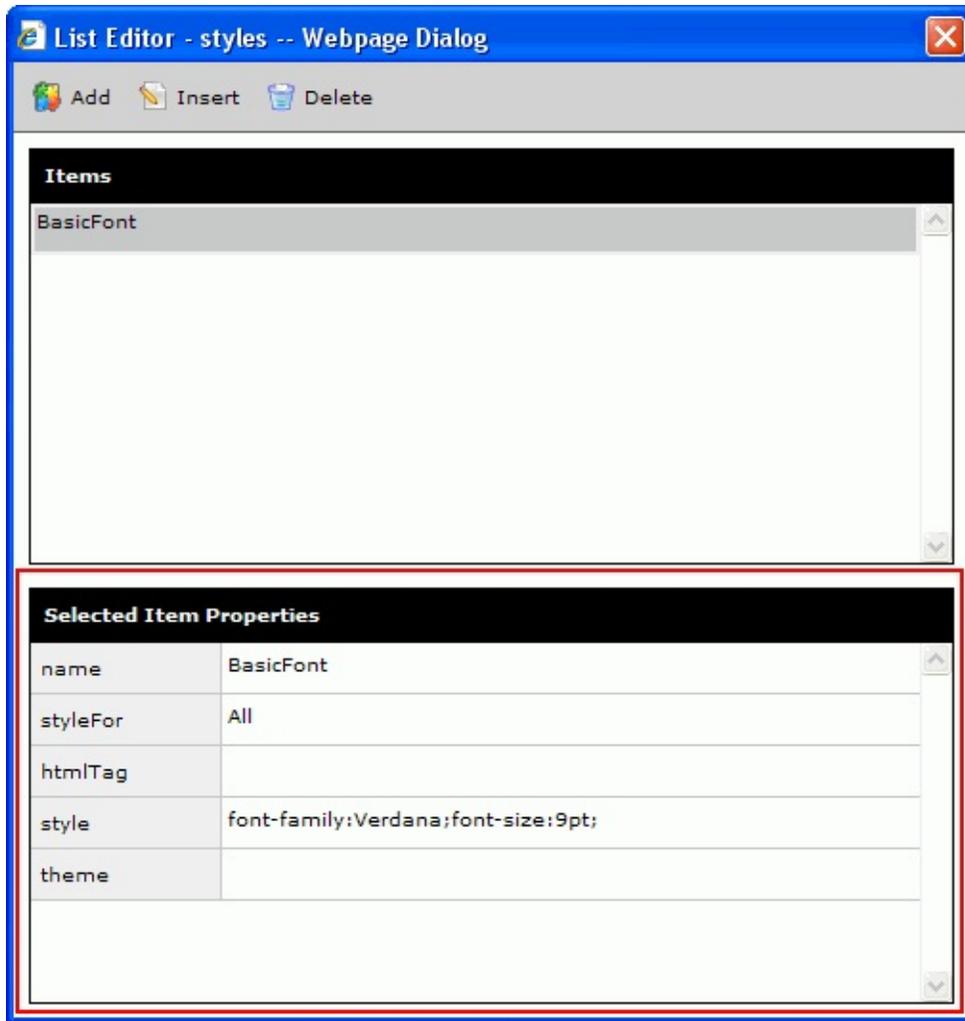


Selected Item Properties	
name	BasicFont
styleFor	All
htmlTag	
style	
theme	

5. Click on the Style property. A window showing all style properties is displayed.
6. Locate the font-family attribute and specify Verdana.
7. Locate the font-size property and specify 9pt:



8. Now double-click Styles in the aXes Designer to view the style you have added:

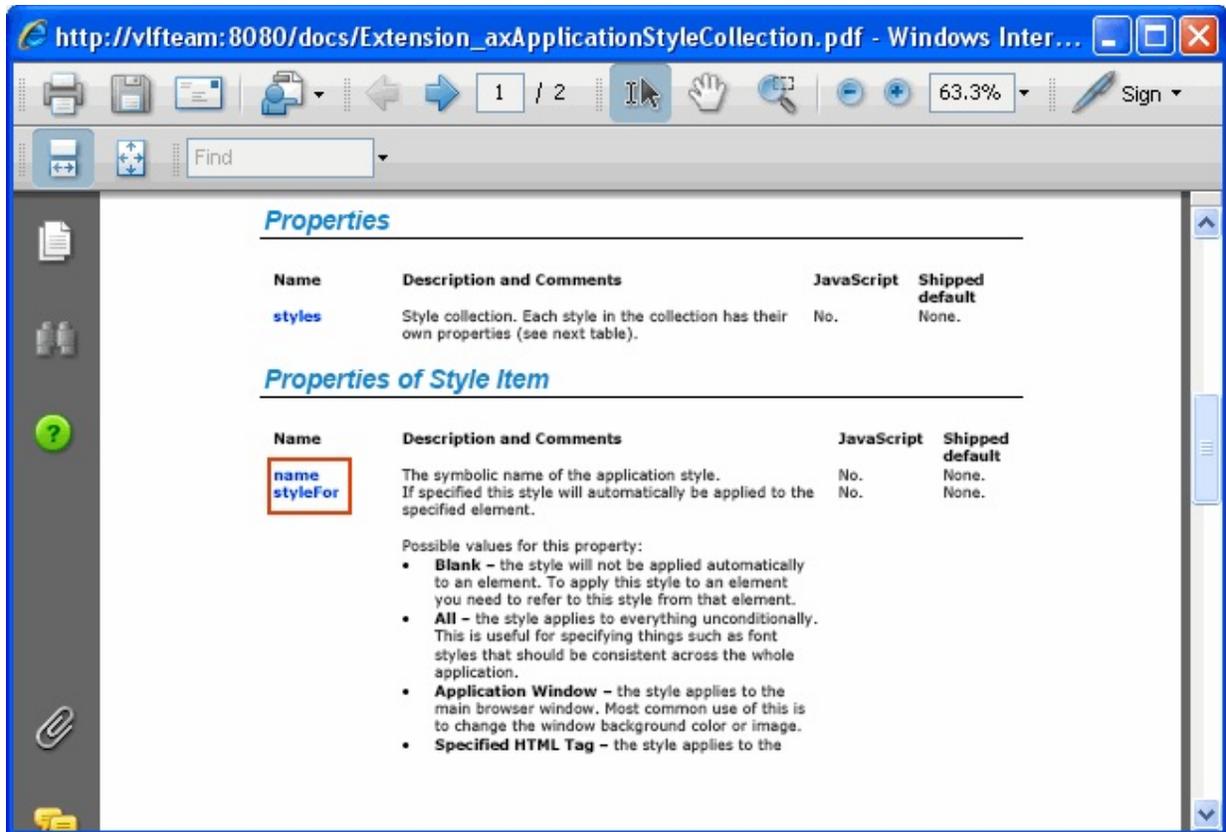


When setting font sizes, please note that the screens shown in a RAMP application are automatically sized to fit the available space. This may mean that the font size displayed may be smaller than what you have specified.

10. To understand what the Style properties are, click on the **i** icon in front of the Styling heading in the aXes Designer window to bring up the context sensitive help:

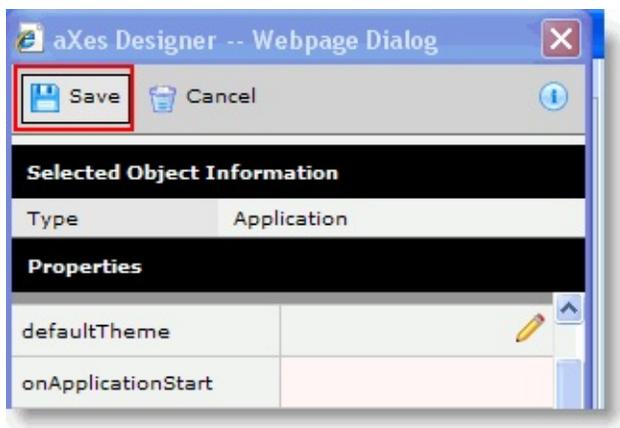


11. Read the description for the StyleFor property:



The value All for the StyleFor property of the BasicFont style means it applies to all text in all screens which have been defined in aXes (when you edit a screen and save it in aXes Designer, a screen_XXXXX.js file is created for it in the aXes screens folder).

12. Click Save on top of the aXes Designer window to save the style you have created:



Note that the font of the screen is now Verdana 9pt:

The screenshot shows a software window with a title bar containing 'Session Display Help' and a zoom level of '100%'. The main content area is titled 'Browse/Maintain Employee and Skill Files' and contains a form for employee information and a table of skills.

Employee Information:

- Employee Number: A1234
- Employee Surname: JACKSON
- Employee Given Name(s): STEPHEN
- Street No and Name: 6 Melissa Place
- Suburb or Town: West Pennant Hills
- State and Country: NSW Australia (Postcode: 2125)
- Home Phone Number: (02) 9874 7773
- Department Code: SD +
- Section Code: ES +
- Start Date (DDMMYY): 14/08/96 +
- Termination Date (DDMMYY): 0/00/00 +

Skills Table:

Date Skil Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
3/05/98	ADMIN2	Administratn Part 2		D
5/05/98	ECD	Economics Degree		D
5/06/94	ENG	English Degree	With distinction	D

Footer: F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

RAMP-TSAD04 Step 3. Hide Repetitive Information

In this step you will hide the Employee Number field on the screen because the employee number is visible in the instance list and the instance bar in the Framework and it cannot be changed on this screen.

When modernizing 5250 screens, you should always consider:

Hiding 5250 screen identifiers and titles. Use Framework tabs and hints instead. Leaving this information on a 5250 screen is a dead giveaway that the underpinning screen is a 5250 screen. Always ask yourself what value the information has. If it has none or little value then hide it.

Hide screen dates and times. The Framework window shows the date and time.

Hide repetitive and key information that is already displayed on the instance bar, instance list or in some other place. Again, question the value of repetitive information in the large framework context.

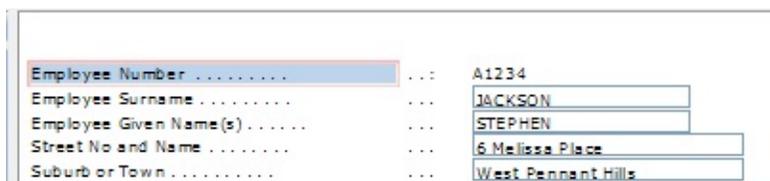
Hide any application version details. The Help menu's About option is the Windows way to do this.

Don't use aXes screen title bars or stripes. They draw the users eye way to information that is often redundant and has no value add.

1. Click on Edit Screen on top of the aXes Designer.



2. Select the Employee number label.



3. Set its Visible property to False.
4. Then select the dots after the label that have become separated from the

label with the font change and set the Visible property to False.

5. Lastly set the Visible property of the Employee Number field to False.
6. Save the screen customization.

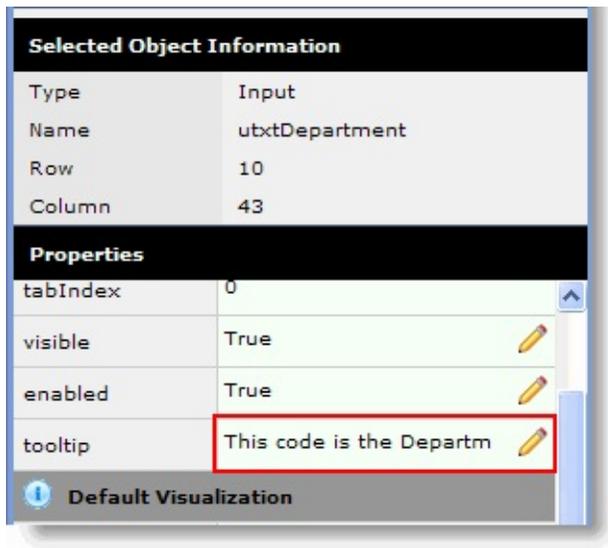
The employee number label and field are no longer visible on the screen.

RAMP-TSAD04 Step 4. Add a Tooltip

In this step you will add a tooltip for the Department Code field. You can use tooltips to provide longer labels and context sensitive help in your screens.

1. Click on Edit Screen in the aXes Designer window.
2. Select the Department Code field.
3. Locate the Tooltip property of the field and copy and paste in this text:

This code is the Department that the employee currently works for. It is often referred to as their "DC" code. Their id badges must always display this code or they may be refused admittance to company premises.



4. Save the screen customization.
5. Test the tooltip by hovering the cursor on the Department Code field:

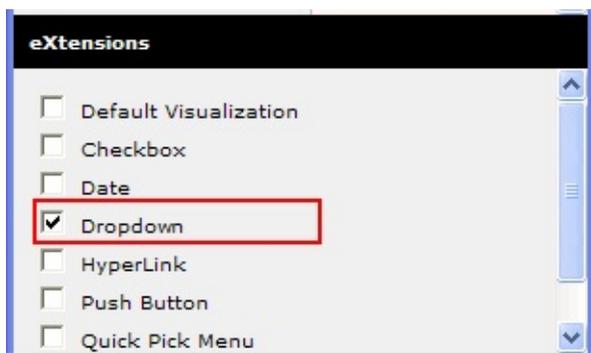
Employee Surname	JACKSON	
Employee Given Name(s)	STEPHEN	
Street No and Name	6 Melissa Place	
Suburb or Town	West Pennant Hills	
State and Country	NSW Australia	2125
Home Phone Number	(02) 9871 7773	
Department Code	SD +	
Section Code	ES	
Start Date (DDMMYY)	14/	
Termination Date (DDMMYY)	0/0	

This code is the Department that the employee currently works for. It is often referred to as their "DC" code. Their id badges must always display this code or they may be refused admittance to company premises.

RAMP-TSAD04 Step 5. Add a Drop-Down

In this step you will make the State and Country field a drop-down which shows states in the US. The data for the drop-down comes from a static table which is shipped with aXes eXtensions.

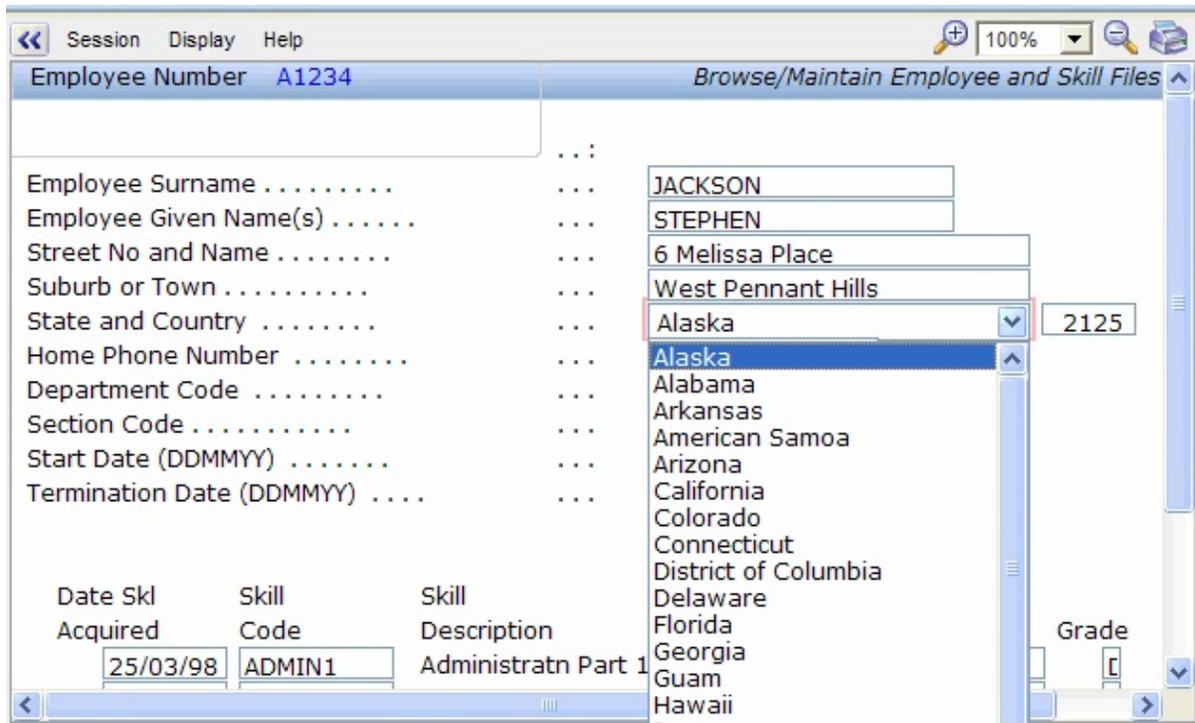
1. Click on Edit Screen in the aXes Designer window.
2. Select the State and Country field on the DisplayEmployee screen.
3. In aXes Designer, change its extension from Default Visualization to Drop-Down:



4. Change the dataSourceType of the drop-down to Static Table.
5. Specify USState as the tableName.



6. Save the screen customization.
7. Test the drop-down:

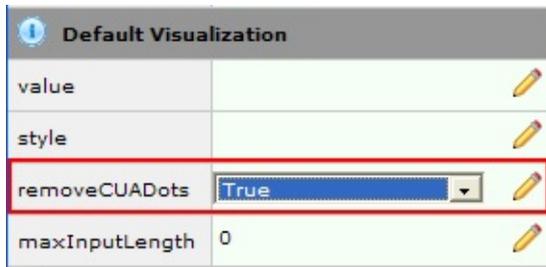


You can set the values displayed in the drop-down also by entering fixed values or by using a dynamic table. The aXes tutorials contain detailed information about how to do this.

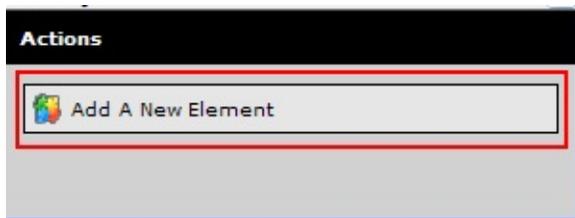
RAMP-TSAD04 Step 6. Organize Fields inside Group Boxes

In this step you will logically organize the fields on the screen by putting them into group boxes.

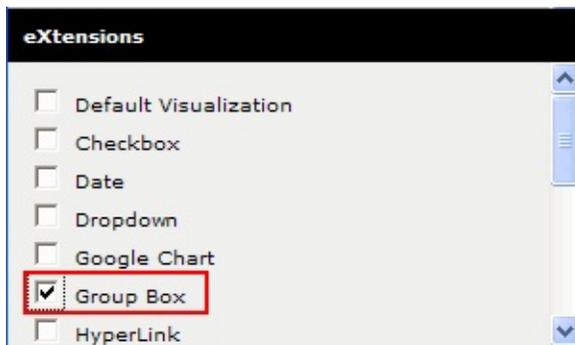
1. Click on Edit Screen in the aXes Designer window.
2. Set the removeCUADots property to True for all elements on the screen that have dots.



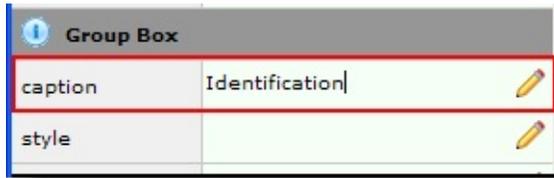
3. Save this change and put the aXes Designer back in edit mode.
2. Add a new element to the screen:



3. Make it a Group Box extension:



4. Make the Caption of the group box Identification.



5. Initially size it like this:



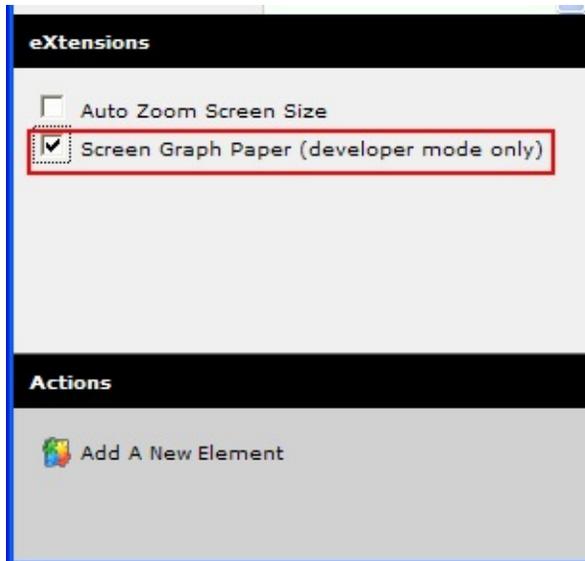
6. Then resize the fields and move them inside the group box like this (leave some room above the group box):



When aligning the fields you may want to use the screen graph paper:

7. Click on an empty space on the screen.

8. Select the Screen Graph Paper option on in the aXes Designer window.



It looks like this:

Identification

Employee Surname: JACKSON

Employee Given Name(s): STEPHEN

Street No and Name: 6 Melissa Place

Suburb or Town: West Pennant Hills

State and Country: [dropdown] 2125

Home Phone Number: (02) 9871 7773

Department Code: SD +

Section Code: ES +

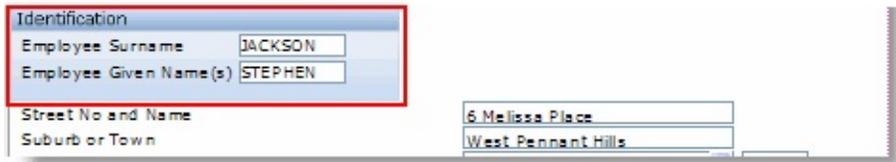
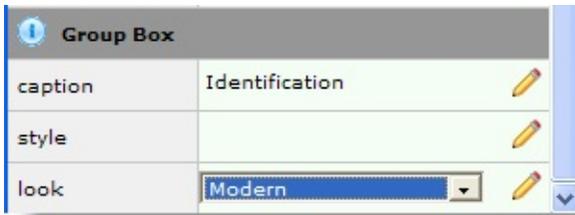
Start Date (DDMMYY): 14/08/96 +

Termination Date (DDMMYY): 01/06/00 +

Date Skil	Skill	Skill	Comment	Grade
Acquired	Code	Description		
25/03/98	ADMIN1	Administrato Part 1	Met requirement	1
3/05/98	ADMIN2	Administrato Part 2		1
5/05/98	ECD	Economics Degree		1
5/06/94	ENG	English Degree	With distinction	1

F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

9. Turn the Screen Graph Paper option off.
10. Set the look property of the group box to Modern:



11. Save this change and put the aXes Designer back in edit mode.
12. Add another group box with the Caption Dates and look Modern.
13. Place it next to the Identification group box.
14. Put the Start Date and Termination dates in the group box like this:



15. Add another group box with the caption Contact and Location and look Modern (again leave some space around the group box so that it will be surrounded by the screen's background color).
16. Place it under the Identification group box and put the remaining fields except for the skills details in it:

Session Display Help

Identification

Employee Surname: JACKSON
Employee Given Name(s): STEPHEN

Dates

Start Date (DDMMYY): 14/08/96 +
Termination Date (DDMMYY): 0/00/00 +

Contact and Location

Street No and Name: 6 Melissa Place
Suburb or Town: West Pennant Hills
State and Country: [dropdown] 2125
Home Phone Number: (02) 9871 7773
Department Code: SD +
Section Code: ES + [Employment Contract](#)

Date Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratrn Part 1	Met requirement	[dropdown]
3/05/98	ADMIN2	Administratrn Part 2		[dropdown]
5/05/98	ECD	Economics Degree		[dropdown]
5/06/94	ENG	English Degree	With distinction	[dropdown] +

F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

17. Finally, add a group box for the skills information with the caption Skills, again leave some room around the group box:

Identification

Employee Surname: JACKSON
Employee Given Name(s): STEPHEN

Dates

Start Date (DDMMYY): 14/08/96 +
Termination Date (DDMMYY): 0/00/00 +

Contact and Location

Street No and Name: 6 Melissa Place
Suburb or Town: West Pennant Hills
State and Country: [dropdown] 2125
Home Phone Number: (02) 9871 7773
Department Code: SD +
Section Code: ES + [Employment Contract](#)

Skills

Date Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratrn Part 1	Met requirement	[dropdown]
3/05/98	ADMIN2	Administratrn Part 2		[dropdown]
5/05/98	ECD	Economics Degree		[dropdown]
5/06/94	ENG	English Degree	With distinction	[dropdown] +

F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

18. Save the screen customization.

RAMP-TSAD04 Step 7. Add Up and Down Buttons to Subfile

In this step you will name replace the plus sign indicating more entries exist in a the subfile with up and down keys.

1. Select the plus indicator for the subfile.

Skill	Description	Comment	Grade
	Administratn Part 1	Met requirement	F
	Administratn Part 2		F
	Advanced Programming		F
	Communications Degre		C +

2. Name it moreindicator on the Screens tab.

ChangeMode

Description:

Subfiles:

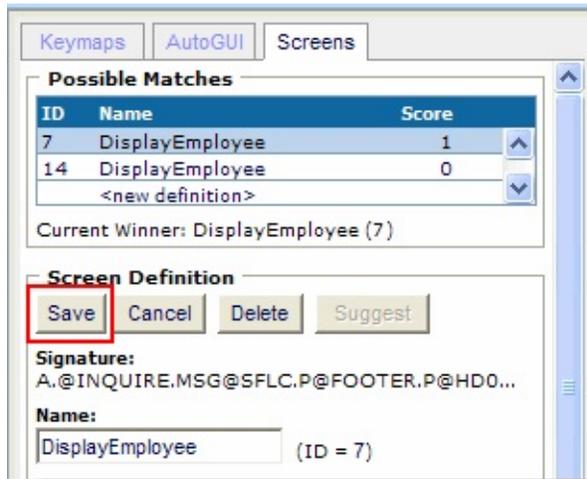
Start	End	Name	Lines/Entry	Head Lines
18	21	R@F00003	1	+

Fields:

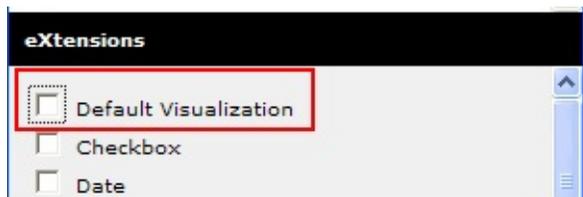
Name	Row	Col	Sfl
	17	70	
Grade	18	7	Y
	18	16	Y
	18	28	Y
Administratn Part 1	18	49	Y
	18	73	Y
moreindicator	21	79	Y
	23	2	

F1=Help F3=Exit F4=Prompt F12=Cancel

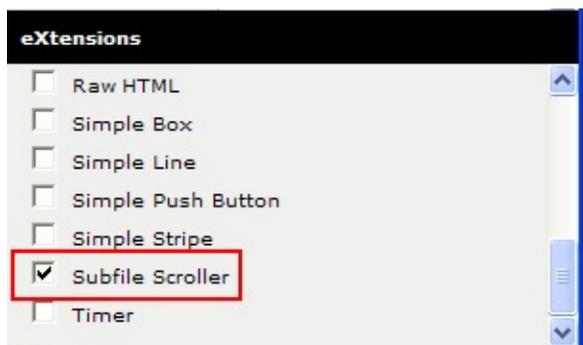
3. Save the screen.



4. In aXes Designer, uncheck the default visualization of the + sign so that it disappears



5. Add a new element, and make its type Subfile Scroller.



6. Specify moreindicator as the markerFieldName.

Subfile Scroller

markerFieldName 

scrollerStyle 

7. Position and size the subfile scroller.
8. Save the screen customization.
9. Do a partial save of the RAMP definition.
10. Test the up and down buttons on the subfile.

Date Acquired	Skl Code	Skill Description	Comment	Grade
30/06/96	MANAGE2	Management Course 2	Met requirement	P
10/12/97	MANAGE3	Management Course 3	Met requirement	P
1/03/98	MANAGE4	Management Course 4	Good effort	P
10/12/95	MARKET1	Marketing Course 1		P

RAMP-TSAD04 Step 8. Hide Function Keys and Add a Picture

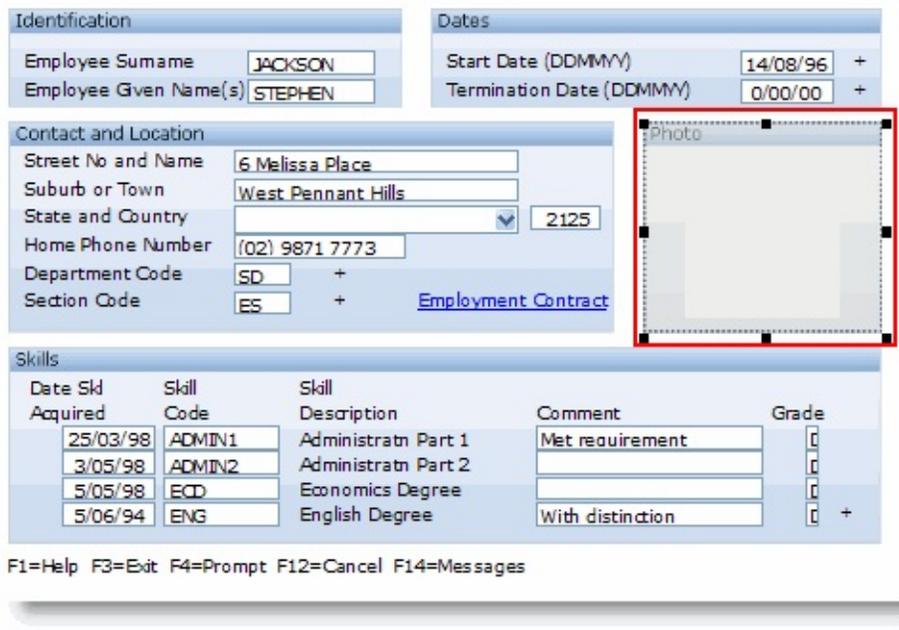
In this step you will hide the function key text shown on the bottom of the screen and you will add a picture of an employee.

When you edit a screen with the aXes Designer, the function key text line on the screen becomes visible, even though it was previously hidden by RAMP. To hide the line:

1. Click on Edit Screen in aXes Designer.
2. Select the function key text line on the bottom of the screen.
3. Set its Visible property to False.

Next you will add an image to the screen:

4. Add a group box with the caption Photo and with Modern look:



The screenshot shows the aXes Designer interface for editing a screen. The form contains several sections: Identification, Dates, Contact and Location, Skills, and a footer with function key shortcuts. A red box highlights a new 'Photo' group box being added to the form.

Identification		Dates	
Employee Surname	JACKSON	Start Date (DDMMYY)	14/08/96 +
Employee Given Name(s)	STEPHEN	Termination Date (DDMMYY)	0/00/00 +

Contact and Location			
Street No and Name	6 Melissa Place		
Suburb or Town	West Pennant Hills		
State and Country	2125		
Home Phone Number	(02) 9871 7773		
Department Code	SD	+	
Section Code	ES	+ Employment Contract	

Skills					
Date Skd Acquired	Skill Code	Skill Description	Comment	Grade	
25/03/98	ADMIN1	Administratn Part 1	Met requirement	[]	
3/05/98	ADMIN2	Administratn Part 2		[]	
5/05/98	ECD	Economics Degree		[]	
5/06/94	ENG	English Degree	With distinction	[]	+

F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

5. Then add a new element and make it an Image extension inside the group box:

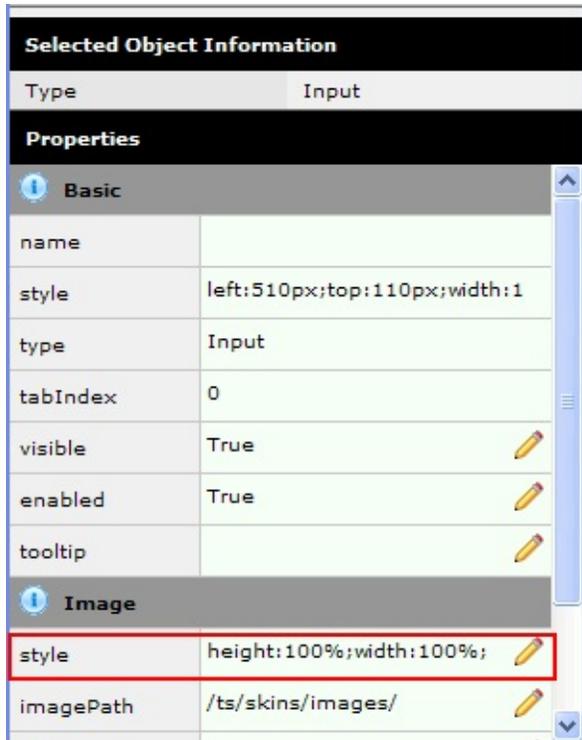
Identification		Dates		
Employee Surname	JACKSON	Start Date (DDMMYY)	14/08/96 +	
Employee Given Name(s)	STEPHEN	Termination Date (DDMMYY)	0/00/00 +	
Contact and Location		Photo		
Street No and Name	6 Melissa Place	Photo		
Suburb or Town	West Pennant Hills			
State and Country	<input type="text"/> 2125			
Home Phone Number	(02) 9871 7773			
Department Code	SD +			
Section Code	ES + Employment Contract			
Skills				
Date Acquired	Skd Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	<input type="text"/>
3/05/98	ADMIN2	Administratn Part 2		<input type="text"/>
5/05/98	ECD	Economics Degree		<input type="text"/>
5/06/94	ENG	English Degree	With distinction	<input type="text"/> +

F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

- Change its imagePath to /ts/skins/images/ and imageName to examplephoto.gif:

Image	
style	
imagePath	/ts/skins/images/
imageName	examplephoto.gif
onClick	

- Click Save. The image is displayed in its full size.
- To shrink it, click on the Style property of the image extension and change the style's height and width to 100% (be careful to select the Style property in the Image group):



The screen will now look like this:

Identification		Dates		
Employee Surname	JACKSON	Start Date (DDMMYY)	14/08/96 +	
Employee Given Name(s)	STEPHEN	Termination Date (DDMMYY)	0/00/00 +	
Contact and Location		Photo		
Street No and Name	6 Melissa Place			
Suburb or Town	West Pennant Hills			
State and Country	[Dropdown] 2125			
Home Phone Number	(02) 9871 7773			
Department Code	SD +			
Section Code	ES + Employment Contract			
Skills				
Date Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	[Dropdown]
3/05/98	ADMIN2	Administratn Part 2		[Dropdown]
5/05/98	ECD	Economics Degree		[Dropdown]
5/06/94	ENG	English Degree	With distinction	[Dropdown] +

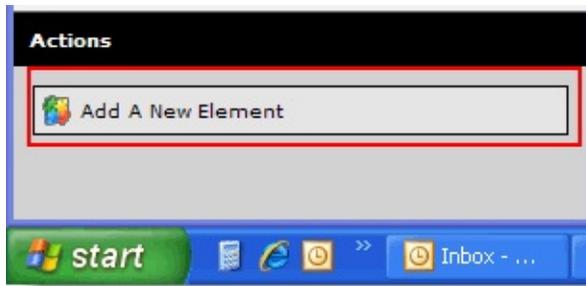
F1=Help F3=Exit F4=Prompt F12=Cancel F14=Messages

9. Save the screen customization

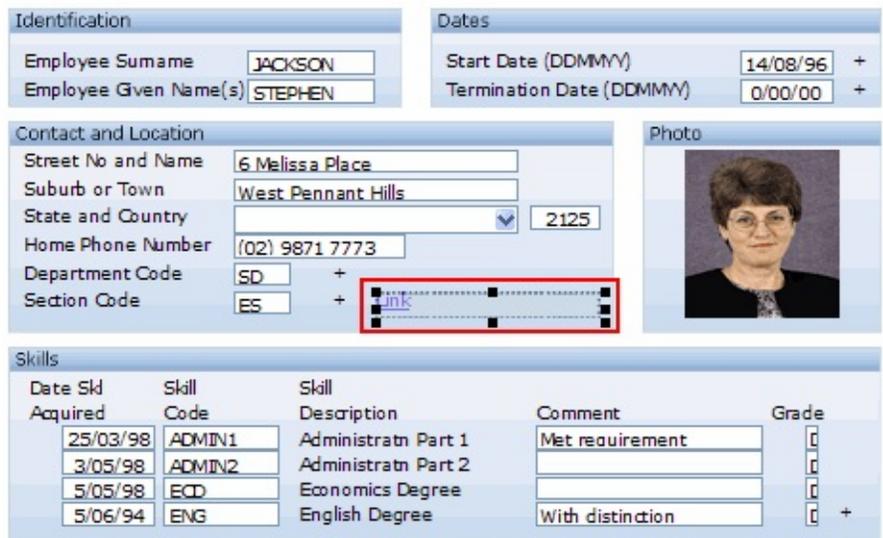
RAMP-TSAD04 Step 9. Add a Hyperlink

In this step you will add a hyperlink to the screen to open a PDF document that resides on the server.

1. Put the screen in edit mode by clicking Edit Screen in the aXes Designer.
2. Add a new element to the screen.



3. Make it Hyperlink extension.
4. Size and position it like this:

A screenshot of an employee record form. The form is divided into several sections: 'Identification', 'Dates', 'Contact and Location', 'Photo', and 'Skills'. The 'Section Code' field in the 'Contact and Location' section contains the text 'ES' followed by a red-bordered box containing a blue hyperlink icon. The 'Skills' section is a table with columns for Date Acquired, Skill Code, Skill Description, Comment, and Grade.

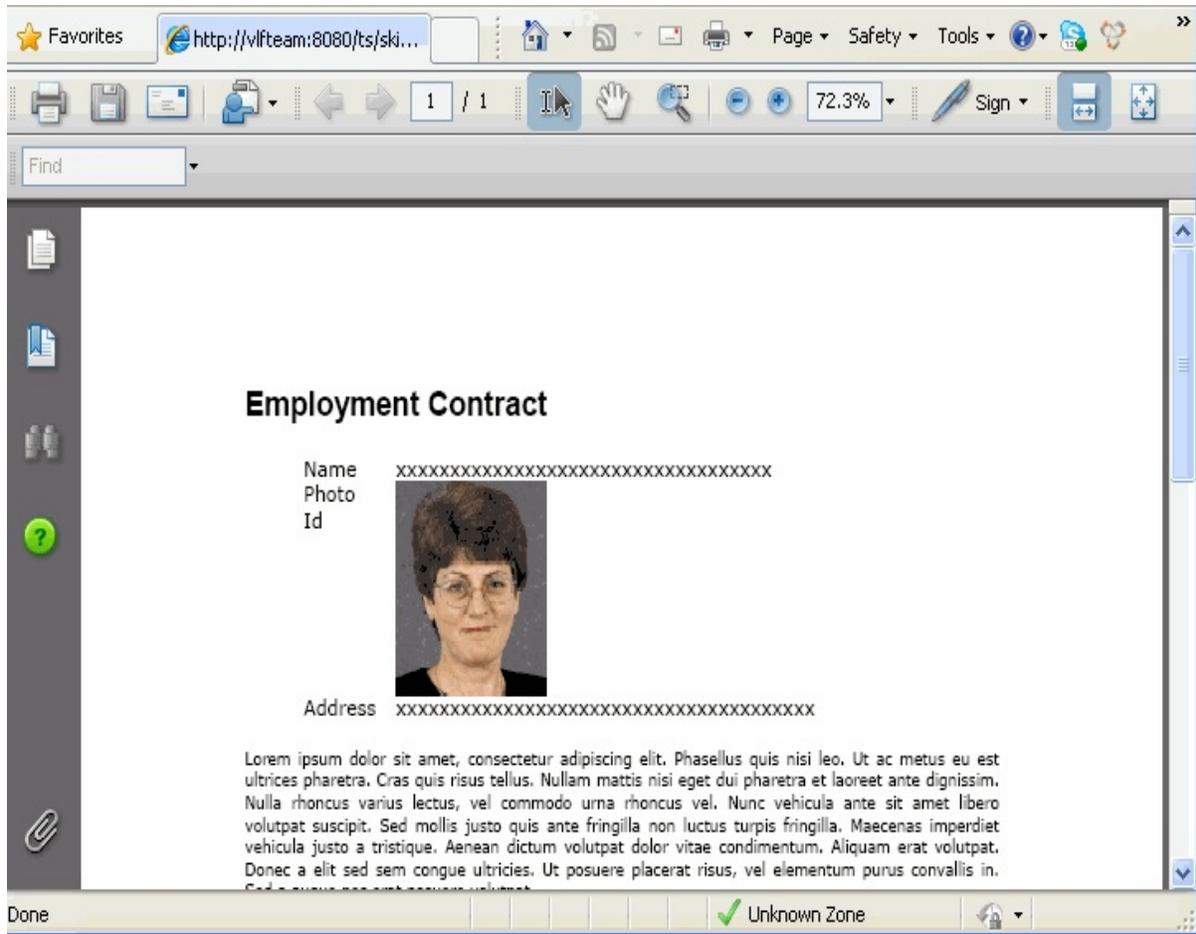
Date Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	[]
3/05/98	ADMIN2	Administratn Part 2		[]
5/05/98	ECD	Economics Degree		[]
5/06/94	ENG	English Degree	With distinction	[] +

5. Set its Caption to Employment Contract.
6. Change the onClick property to:

```
window.open('/ts/skins/images/examplecontract.pdf', '_blank');
```

7. Save the screen customization.
8. Click on the Employment Contract hyperlink on the screen.

A browser window containing the contract PDF on the server is shown:



RAMP-TSAD04 Step 10. Test the Redesigned Screen

In this step you will test the redesigned screen and make sure it works with the Framework themes.

1. Log off in the RAMP-TS 5250 emulator.
2. Restart the Framework to restart aXes.
3. Display employee Veronica Brown in the Personnel application.

The Details command handler now looks like this:

Date Skd Acquired	Skill Code	Skill Description	Comment	Grade
30/06/96	ADMIN1	Administratn Part 1	Met requirement	E
1/03/98	ADMIN2	Administratn Part 2		E
10/12/95	ADVPGM	Advanced Programming		E
4/05/98	COM	Communications Degre		C +

4. Now change the theme of the Framework using the Overall Theme option of the Windows menu to check what the customization looks like with different themes.

This is what the screen looks like with 2007 Olive theme:

Details
 All Details
 Details Wrapper
 Name
 Address
 HR Details
 Events
 Documents

Identification

Employee Surname:

Employee Given Name(s):

Dates

Start Date (DDMMYY):

Termination Date (DDMMYY):

Save

Prompt

Copy to Clipboard

Show Menu Bar

Show Snapshot

Documentation

Turn Trace On

Probe Screen

Contact and Location

Street No and Name:

Suburb or Town:

State and Country:

Home Phone Number:

Department Code: +

Section Code: + [Link](#)

Photo



Skills

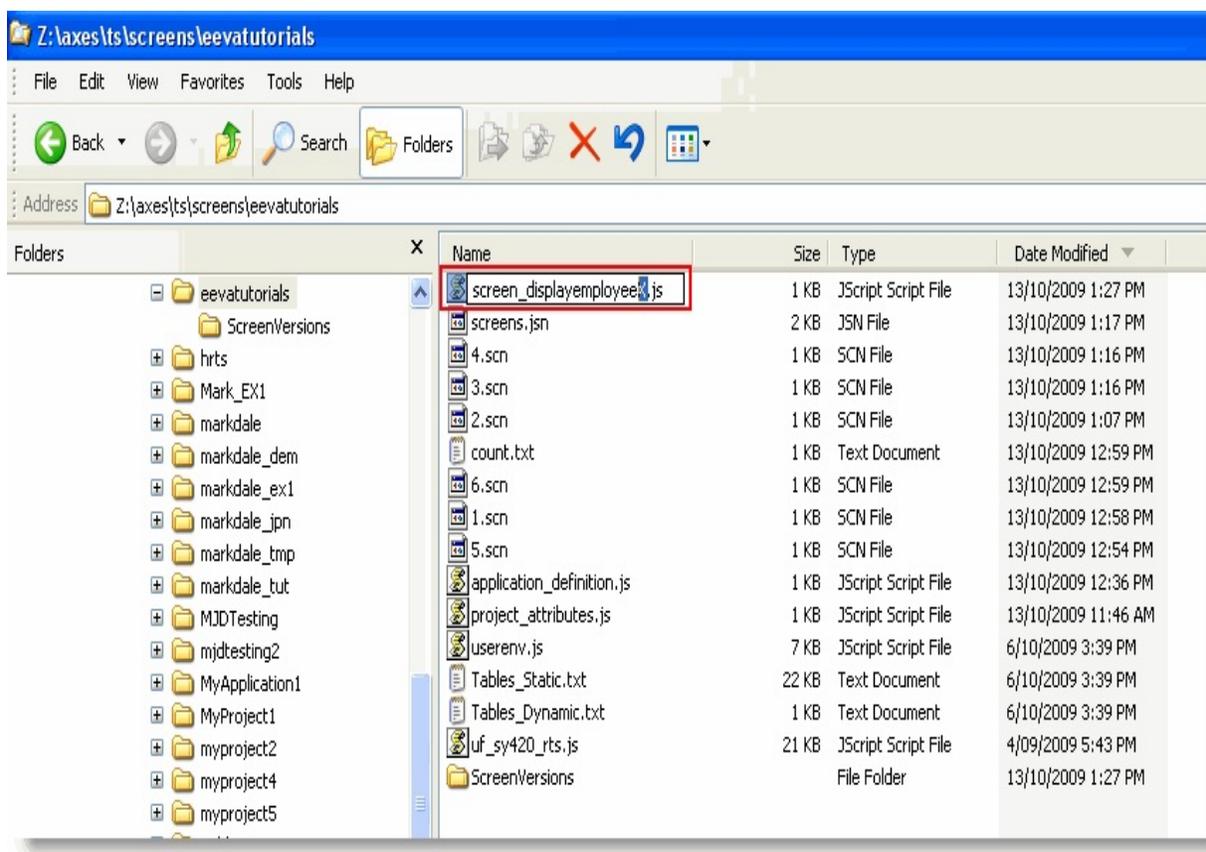
Date Skd Acquired	Skill Code	Skill Description	Comment	Grade
<input type="text" value="30/06/96"/>	<input type="text" value="ADMIN1"/>	Administratn Part 1	<input type="text" value="Met requirement"/>	<input type="text" value="B"/>
<input type="text" value="1/03/98"/>	<input type="text" value="ADMIN2"/>	Administratn Part 2	<input type="text"/>	<input type="text" value="B"/>
<input type="text" value="10/12/95"/>	<input type="text" value="ADVPGM"/>	Advanced Programming	<input type="text"/>	<input type="text" value="B"/>
<input type="text" value="4/05/98"/>	<input type="text" value="COM"/>	Communications Degre	<input type="text"/>	<input type="text" value="C"/> +

Note that here the aXes group boxes are all using a bluish color gradient. They can also be modified to follow the VL theme. Refer to the aXes USERENV file for details of how the color gradients are set up for group boxes. Refer to Axes Tutorial 10 for details of how to find out the current VL theme in aXes scripts.

RAMP-TSAD04 Step 11. Remove the Screen Customization

In this step you will remove the aXes screen customization by renaming the screen_xxxx.js file. You can later reapply the extensions by renaming the file back.

1. Shut down the Framework.
2. In Windows Explorer, locate the axes\ts\screens\ directory (you will need to be mapped to your server's IFS drive) or your private definition folder if you are using one.
4. Then locate the screen_displayemployee.js file and rename it screen_displayemployeeX.js.



5. Restart the Framework.
6. Verify that your screen customization has been removed.

When you want to reapply the extensions, remove the X from the file name.

Note also that every time you click Save in the aXes Designer, a copy of the customized screen is stored in the subfolder ScreenVersions in the screens directory or your private definition folder with a name like Screen_XXXXXXXX_YYYYMMDD_HHMMSS_mmmmmm.js.

You can revert to an earlier design by locating the version you want, deleting the existing Screen_XXXXXXXX.js file, and then copying the screen version file into your definition set folder. Rename it to Screen_XXXXXXXX.js.

Remember to end all aXes developer sessions before doing this.

Summary

If you are planning to use aXes to redesign your screens, you should complete the aXes tutorials which can be accessed from the aXes home page.

To launch aXes from your browser, use this url:

<http://hostname:80/wba/home.html> (replace hostname with the name of your host, and if necessary change the default port 80). Click on the **Tutorials** link on the right of the aXes home page and then the **Tutorial 0 – Getting Started** link.

Important Observations

There are sometimes alternative ways of creating a screen enhancement. For example, you can visualize a date field as a calendar drop-down either by using a special field handler or by making it a Date extension in aXes.

You can use aXes from the aXes home page to name and redesign screens without using RAMP Tools. You need to use RAMP Tools for classifying screens and scripting.

Tips

aXes also supports themes. They can directly map to VLF themes. In a nutshell this means that when you define a role-based style in aXes named "KeyDetails" (say) that says key text should be emphasized and bolded - you can actually theme the style so that for VLF theme Blue the text color is dark blue, for theme Silver the text color is black, for theme Olive the text color is dark green, etc.

Don't use bright primary colors and large or fancy fonts. Using large fonts, fancy fonts and bright primary colors may work in web page displayed enhanced 5250 screens. However, the VLF tends to produce more low key screens - so these types of things will stand out unnecessarily and often quite badly.

What You Should Know

How to use aXes eXtensions to enhance the screens in your RAMP application.

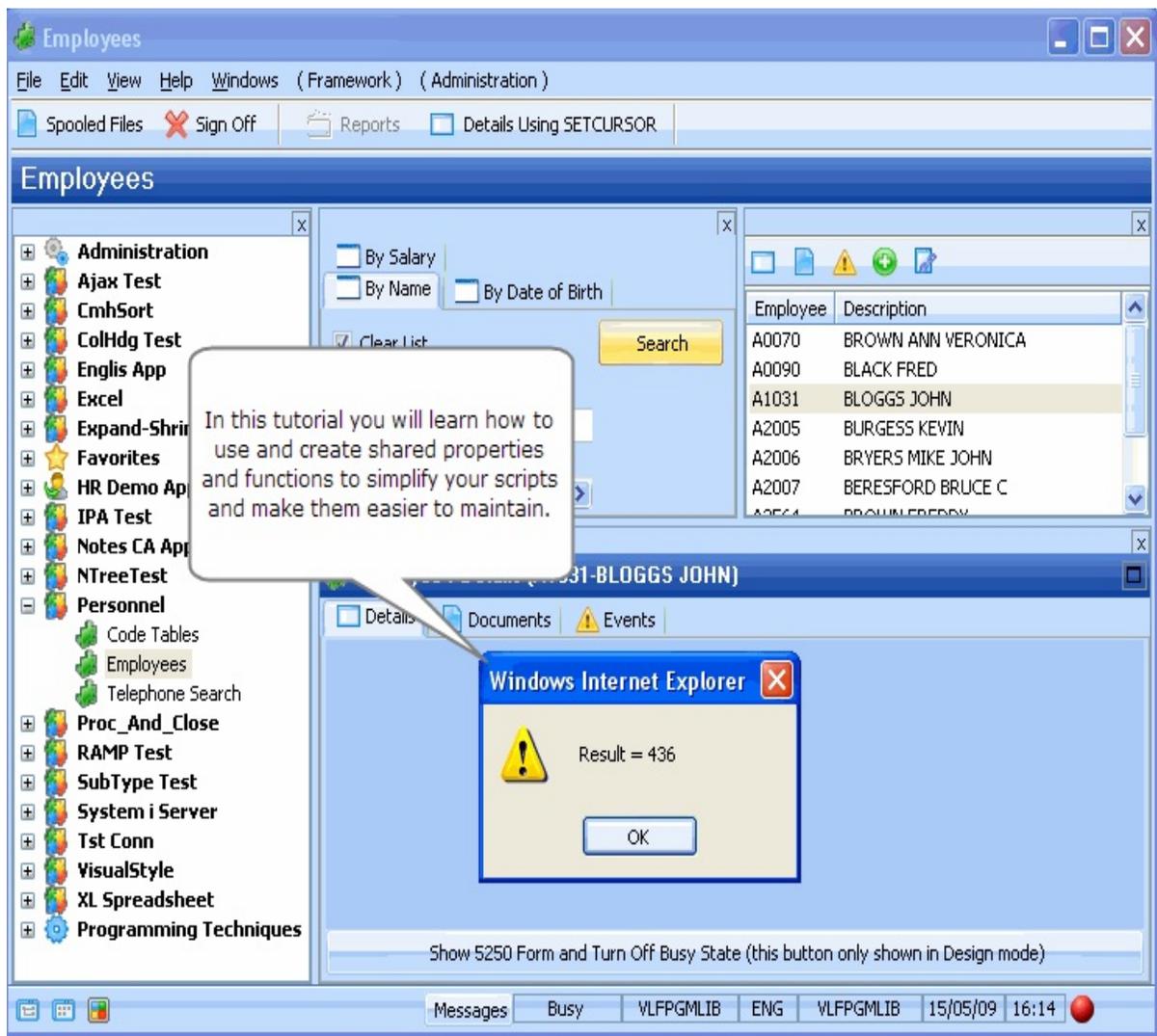
-

RAMP-TSAD05: Using SHARED Properties and Functions

Once you start RAMP scripting you will see patterns and repetitions in your logic. By moving this logic into shared object you can invoke this logic from any RAMP script. The reuse and maintenance benefits of using this feature are obvious.

Objectives

Learn how to use shared properties and functions



To achieve this objective, you will complete the following steps:

Read [What are Shared Scripts?](#)

[RAMP-TSAD05 Step 1. Optional - Creating Your Own Copy of the Shared](#)

Scripts File

[RAMP-TSAD05 Step 2. Accessing SHARED properties and functions](#)

[RAMP-TSAD05 Step 3. Creating your own SHARED properties](#)

[RAMP-TSAD05 Step 4. Creating your own SHARED functions](#)

[Summary](#)

Before You Begin

In order to complete this tutorial, you must have completed the core tutorials RAMP-TS001 - RAMP-TS015.

What are Shared Scripts?

The shared scripts file `uf_sy420_rts.js` can be used to store common JavaScript properties and functions that can be accessed from all your 5250 screen scripts. This file is normally stored in the RAMP-TS skins folder. However, a private version of the file can also reside in the nominated Private Definitions Folder (see [RAMP-TSAD05 Step 1. Optional - Creating Your Own Copy of the Shared Scripts File](#)).

To see what the shared scripts file is like:

1. In Windows Explorer set up a mapped drive so that you can access folder `\axes\ts\skins`.
2. Using Notepad or a text editor locate file `uf_sy420_rts.js` and open it. It looks like this:

```
/* =====  
/* Note that this file is used when using RAMP-  
TS as the RAMP 5250 server      */  
/* =====  
  
/* This file is for common JavaScript properties and functions you want to acc  
/* from all your 5250 screen scripts. To provide an unlimited name space your  
/* properties and functions MUST be encapsulated inside an object named SHL  
/* Typically is reside in the \axes\ts\skins folder      */  
  
/* ----- */  
/* The SHARED object contains all customer defined shared scripts and prope  
/* ----- */  
  
var SHARED =  
{  
  
    /* ----- */  
    /* Properties defined as part of the shared object */  
    /* ----- */  
  
    myProperty1 : "a",  
    myProperty2 : 42,
```

```

/* ----- */
/* Functions defined as part of the shared object */
/* ----- */

/* myFunction1 is a test function */

myFunction1 : function(a,b,c)
{
    alert("myFunction1 executed with parameters " + a.toString() + " " + b.toSt
return;
}, /* <===== Note the comma ===== */

/* myFunction2 is another test function */

myFunction2 : function(a,b)
{
    var sResult = "myFunction2 was executed with parameters " + a.toString()
return(sResult);
}, /* <===== Note the comma ===== */

/* Dummy last property that does not have a comma, leave here. All precede

myEndProperty : true

}; /* End of SHARED object definition */

```

The structure of this file is simple:

The line `var SHARED =` defines the start of a JavaScript object named SHARED (you must use the name SHARED).

Within the SHARED object are 2 properties named `myProperty1` and `myProperty2`.

There are also 2 functions called `myFunction1` and `myFunction2` that receive 3 and 2 parameters respectively.

These properties and function serve no purpose other than to demonstrate how they are defined inside the SHARED object. Note especially the comments indicating the use of commas to separate the functions.

This object format is pure JavaScript. It is not unique to RAMP.

By using this technique you will create a preserved namespace for your code that will never conflict with anything else.

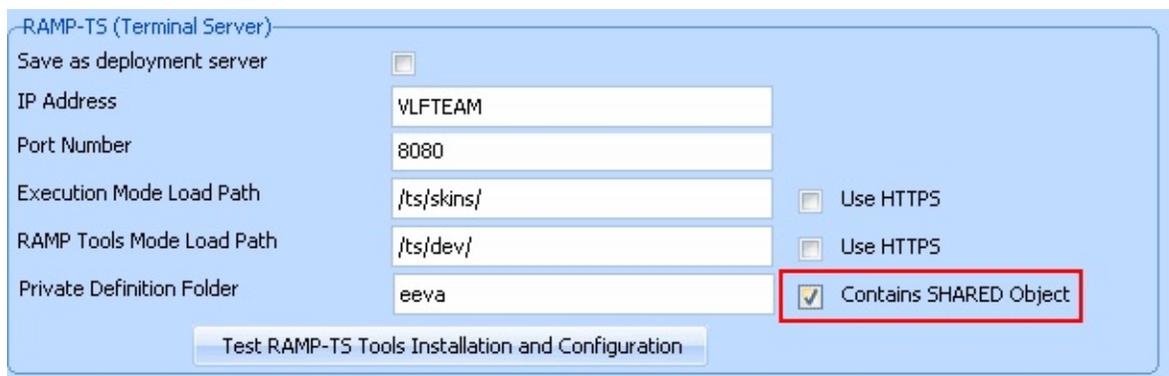
RAMP-TSAD05 Step 1. Optional - Creating Your Own Copy of the Shared Scripts File

Shared scripts are normally stored in the RAMP-TS skins folder in a file named UF_SY420_RTS.JS. However, you can make a private version of the SHARED scripts file in your Private Definition Folder.

Note: If you are completing these tutorials in a classroom setting, there will be a Private Definition Folder set up for every user. However, in a real project you should never do this. Projects should be set up on a discrete project basis because work done in a project folder cannot be merged with work done in another project folder. It is normal for multiple developers to be working on the same project with the same definition set.

To create a private version of the shared scripts file in your Private Definition Folder:

1. In Windows Explorer set up a mapped drive so that you can access the folder `\axes\ts\skins\`.
2. Copy `uf_sy420_rts.js` and paste it into your private definition folder `\axes\ts\screens\MyPrivateDefinitionFolder`
3. In the Server Details in the Framework, check the [Contains SHARED Object](#) option in the RAMP-TS (Terminal Server) group box to indicate to RAMP-TS that the SHARED Object file is in the Private Definitions Folder.



The screenshot shows the 'RAMP-TS (Terminal Server)' configuration window. It contains several fields and checkboxes:

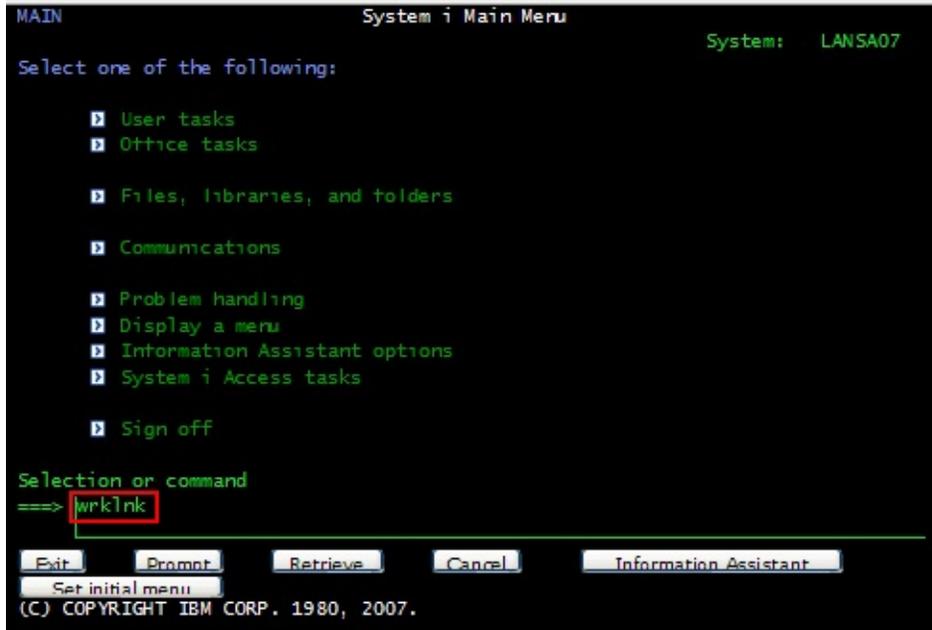
- Save as deployment server:
- IP Address: VLFTEAM
- Port Number: 8080
- Execution Mode Load Path: /ts/skins/
- RAMP Tools Mode Load Path: /ts/dev/
- Private Definition Folder: eeve
- Use HTTPS (Execution Mode):
- Use HTTPS (RAMP Tools Mode):
- Contains SHARED Object: (highlighted with a red box)

At the bottom, there is a button labeled 'Test RAMP-TS Tools Installation and Configuration'.

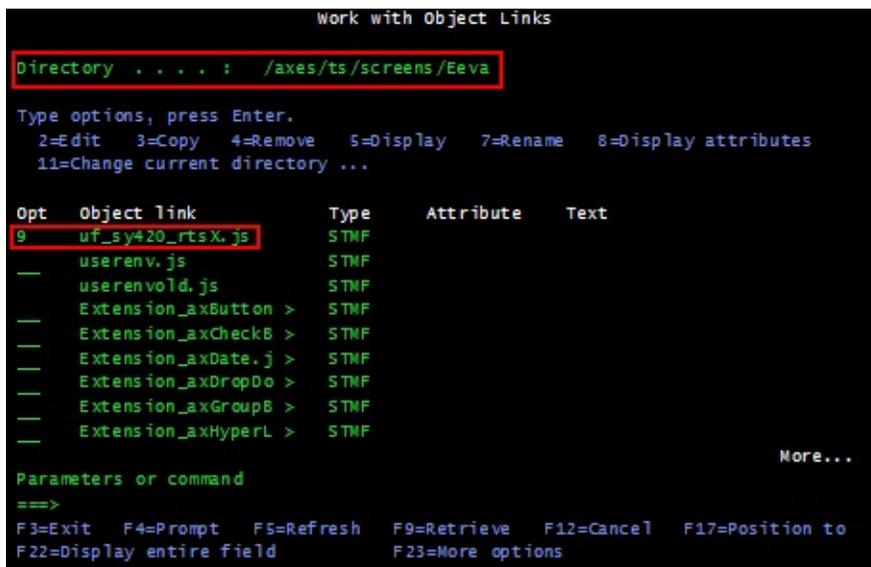
4. Save the Framework.

In order for RAMP-TS to recognize the file, you must ensure that it only has *PUBLIC *R user authority. To check this:

5. In your IBM i, use the WRKLNK command:



6. Navigate to your private definition folder and view the authorities of UF_SY420_RTS.JS.



7. Ensure that *PUBLIC user only has *R authority:

```
Work with Authority

Object . . . . . : /axes/ts/screens/Eeva/uf_sy420_r
Type . . . . . : STMF
Owner . . . . . : VLFP@MLIB
Primary group . . . . . : AXES_GROUP
Authorization list . . . . . : *NONE

Type options, press Enter.
 1=Add user  2=Change user authority  4=Remove user

Opt  User          Data          --Object Authorities--
     User          Authority  Exist  Mgt  Alter  Ref
--
_   *PUBLIC        *R          X      X    X     X
_   VLFP@MLIB     *RWX       X      X    X     X
_   AXES_GROUP    *RWX       X      X    X     X
_   AXES          *RWX       X      X    X     X

Parameters or command
===>
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve
F11=Display detail data authorities  F12=Cancel  F24=More keys
(C) COPYRIGHT IBM CORP. 1980, 2007.
```

You are now ready to start using your private copy of the shared scripts file.

RAMP-TSAD05 Step 2. Accessing SHARED properties and functions

In this step you will learn how to access shared properties and functions.

1. Locate the two test properties in your shared scripts file to see what they are:

```
/* ----- */
/* The SHARED object contains all customer defined shared scripts and properties */
/* ----- */
var SHARED =
{
    /* ----- */
    /* Properties defined as part of the shared object */
    /* ----- */
    myProperty1 : "a",
    myProperty2 : 42,
```

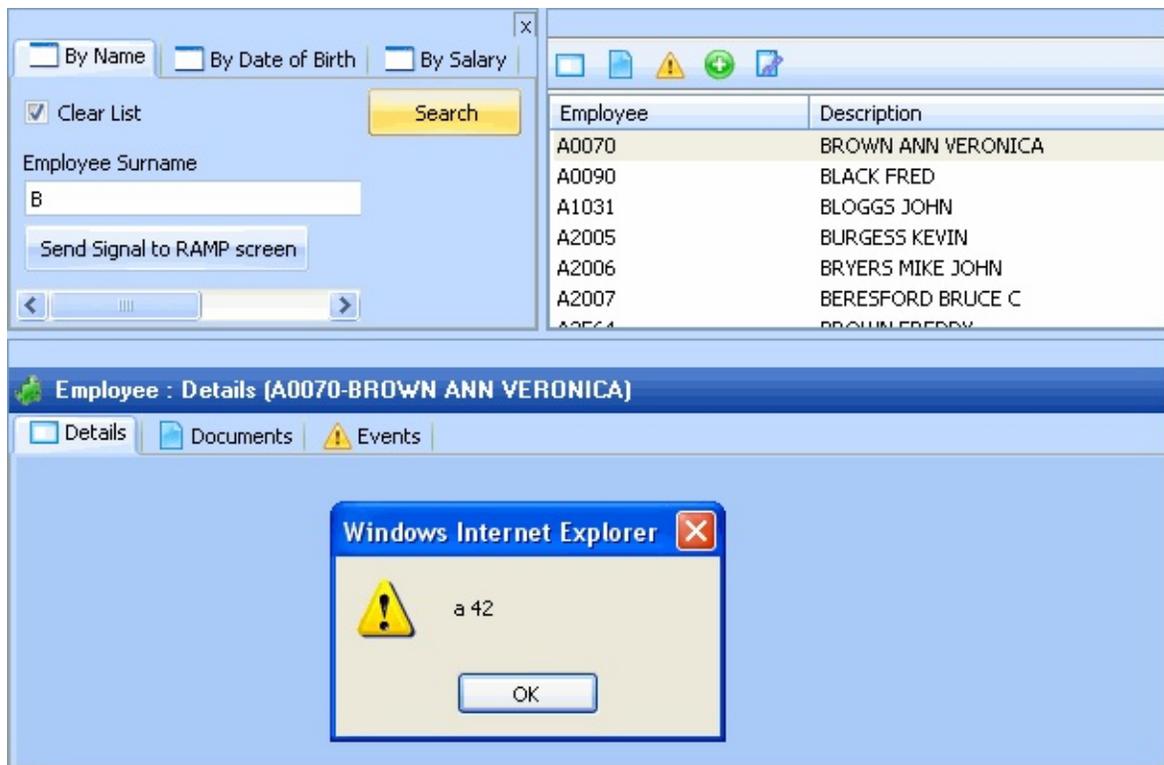
2. Open the RAMP script for the DisplayEmployee screen and add this code to the vHandle_ARRIVE function to display the values of the properties:

```
alert(SHARED.myProperty1 + " " + SHARED.myProperty2);
```

Your code will look like this:

```
else
{
    window.alert(SHARED.myProperty1 + " " + SHARED.myProperty2);
    SHOW_CURRENT_FORM(true); /* Show the form in the framework and show VLF buttons */
    GET_FORM_MESSAGE(22); /* Extract messages and hide the message line */
    HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons displayed */
    SETBUSY(false); /* Last thing done - turn off the busy state */
}
/* <ARRIVE /> - Do not remove or alter this line */
```

3. Commit changes, do a partial save and then display the details of an employee so as to execute the arrival script. You will see a message box like this appear:



The message box is displaying the values of the properties SHARED.myProperty1 and SHARED.myProperty2.

4. Locate the test function Function1 in your shared scripts file to see what it does:

```

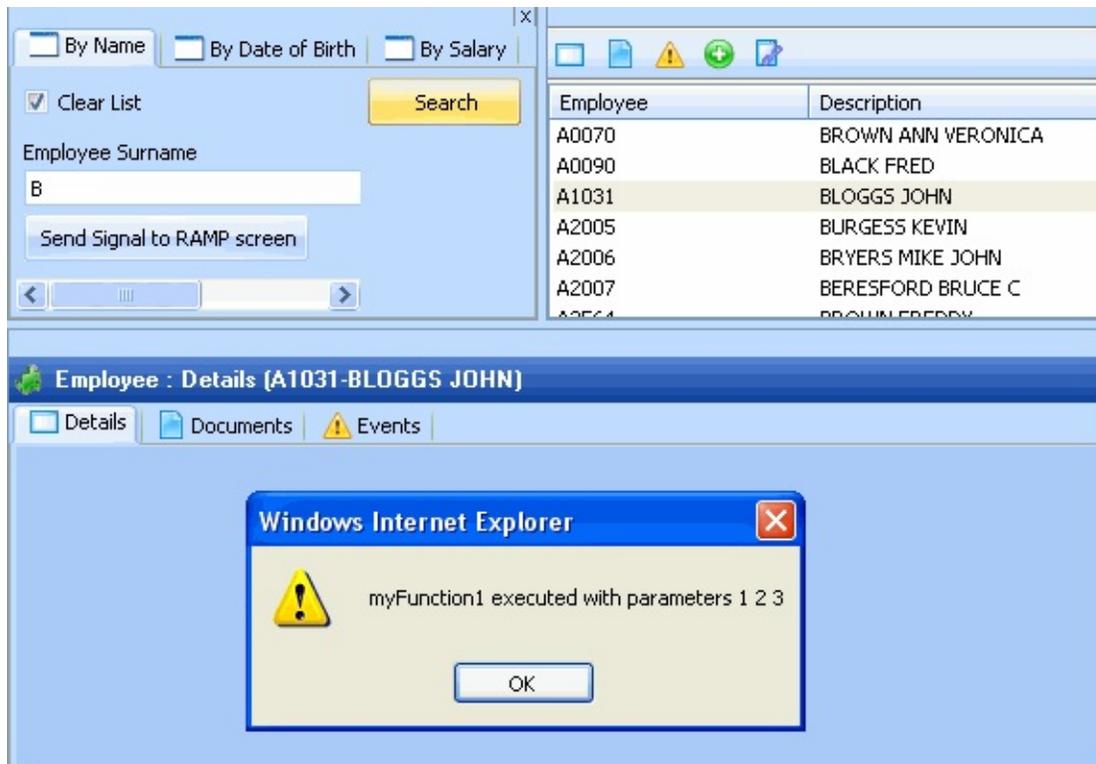
/* ----- */
/* Functions defined as part of the shared object */
/* ----- */
myFunction1 : function(a,b,c)
{
  window.alert("myFunction1 executed with parameters " + a.toString() + " " + b.toString() + " "
+ c.toString() );
  return;
}, /* <===== Note the comma ===== */

```

5. Now add this code to the arrival script:

```
SHARED.myFunction1("1",2,"3");
```

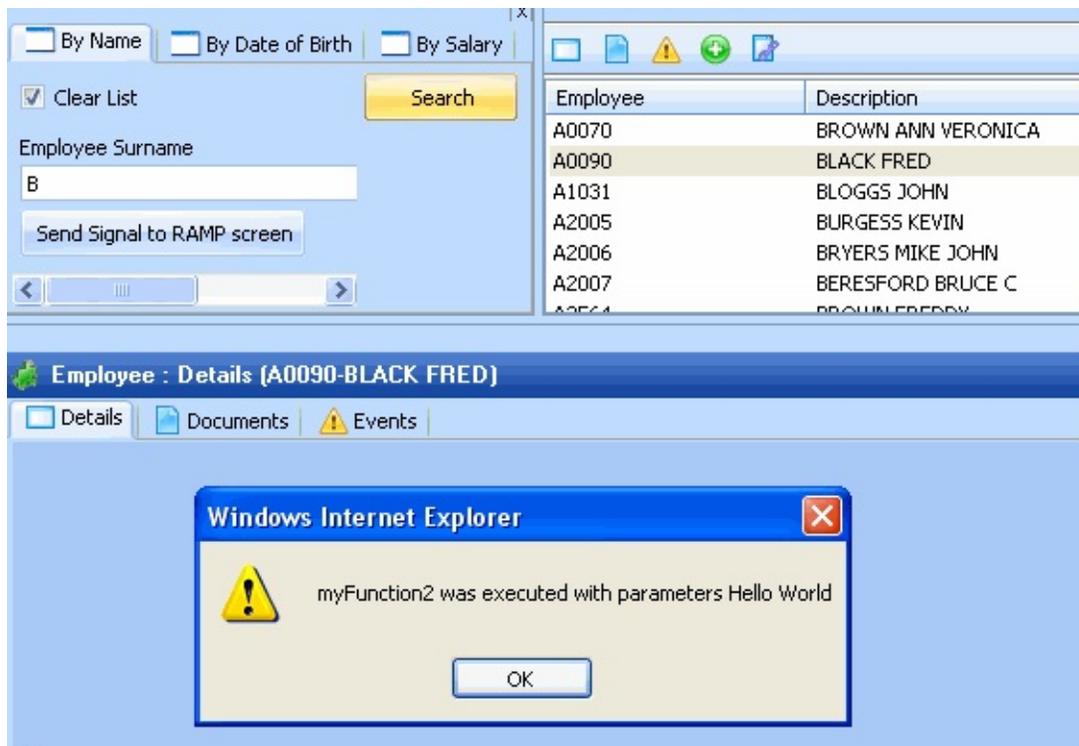
- Commit changes, do a partial save and then display the details of another employee. You will see another message box appear like this, indicating you have executed function myFunction1 in the SHARED object:



- Finally, add this code to your RAMP script:

```
var sMessage = SHARED.myFunction2("Hello","World");  
alert(sMessage);
```

You will see another message box appear like this:



You now know how to access shared properties and shared logic defined in the SHARED object.

RAMP-TSAD05 Step 3. Creating your own SHARED properties

1. Modify file uf_sy420_rts.js using Notepad by defining a new property named MessageLineNumber:

```
messageLineNumber : 22,
```

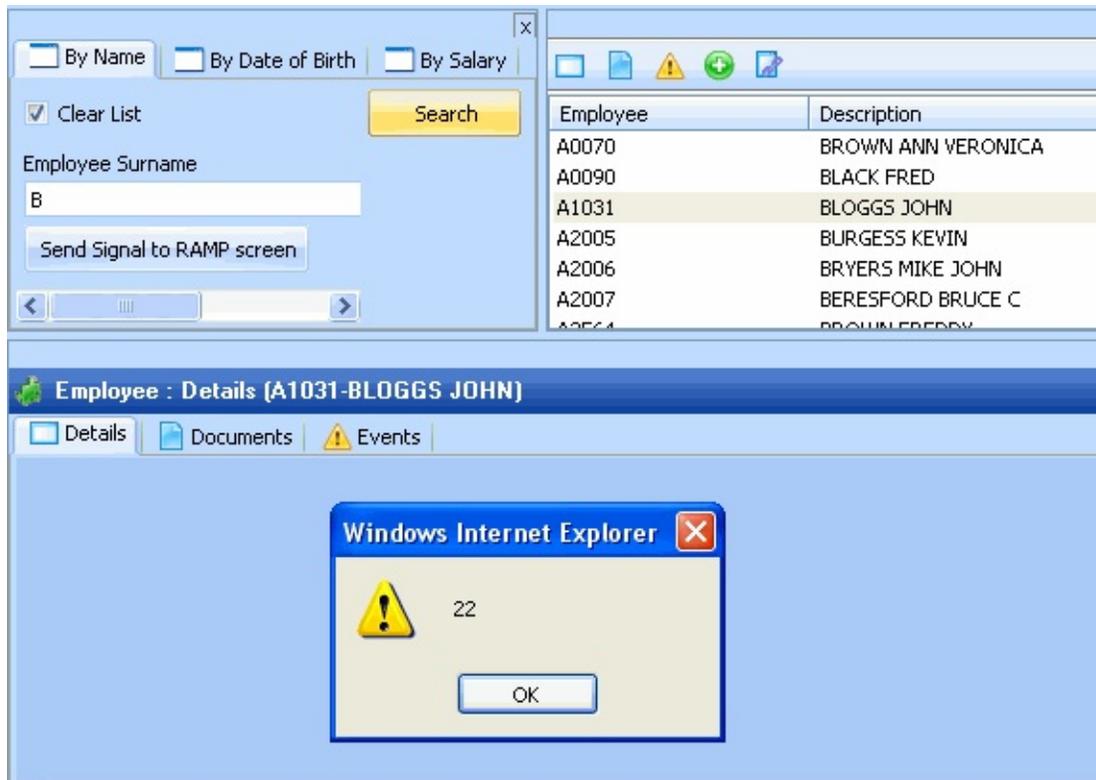
Your code will look like this:

```
/* ----- */
/* The SHARED object contains all customer defined shared scripts and properties */
/* ----- */
var SHARED =
{
    /* ----- */
    /* Properties defined as part of the shared object */
    /* ----- */
    myProperty1 : "a",
    myProperty2 : 42,
    messageLineNumber : 22,
    /* ----- */
    /* Functions defined as part of the shared object */
    /* ----- */
}
```

2. Save and restart the Framework.
3. Now in the vHandle_ARRIVE function of the DisplayEmployee screen's script, remove the code you added in the preceding steps and add this line of code:

```
alert(SHARED.messageLineNumber);
```

4. Execute the DisplayEmployee screen. You will see a message box like this appear.



SHARED properties like this are useful for centralizing definitions, making them easy to change. For example, when using the RAMP function to extract details from your screen, using `SHARED.messageLineNumber` would be better than using the literal 22 (say).

RAMP-TSAD05 Step 4. Creating your own SHARED functions

1. Modify the file `uf_sy420_rts.js` by adding a function named `Add` to it like this:

```
/* Add adds 3 numbers together */  
  
Add : function(a,b,c)  
{  
    return(a + b + c);  
}, /* <===== Note the comma ===== */
```

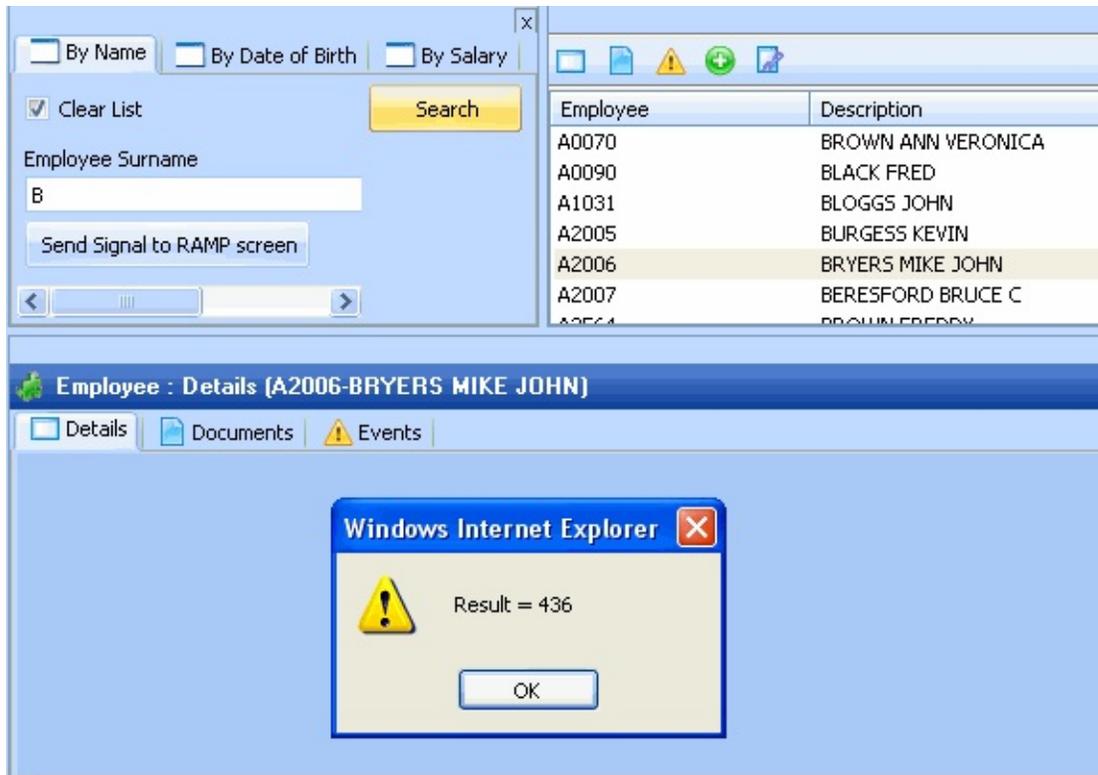
Your code will look like this:

```
/* ----- */  
/* Functions defined as part of the shared object */  
/* ----- */  
  
/* Add adds 3 numbers together */  
Add : function(a,b,c)  
{  
    return(a + b + c);  
}, /* <===== Note the comma ===== */
```

2. Now in your RAMP destination screen script, remove the code from the preceding steps and add these lines of code:

```
var iResult = SHARED.Add(100,200,136);  
alert("Result = " + iResult);
```

3. Close and restart the Framework.
4. Execute your RAMP 5250 destination screen. You should see a message box



That's it. You have passed arguments to your "Add" function and received back its result. Knowing how to do this is all you need to do to start sharing script logic across all your RAMP scripts.

Summary

Important Observations

JavaScript is a very powerful programming language. The more you understand its capabilities the more you can leverage them in your day to day work.

JavaScript knowledge, in an AJAX WEB 2.0 world, is also an increasingly essential IT skill.

There are many free online courses that offer JavaScript training. For example, see <http://www.w3schools.com/>

What You Should Know

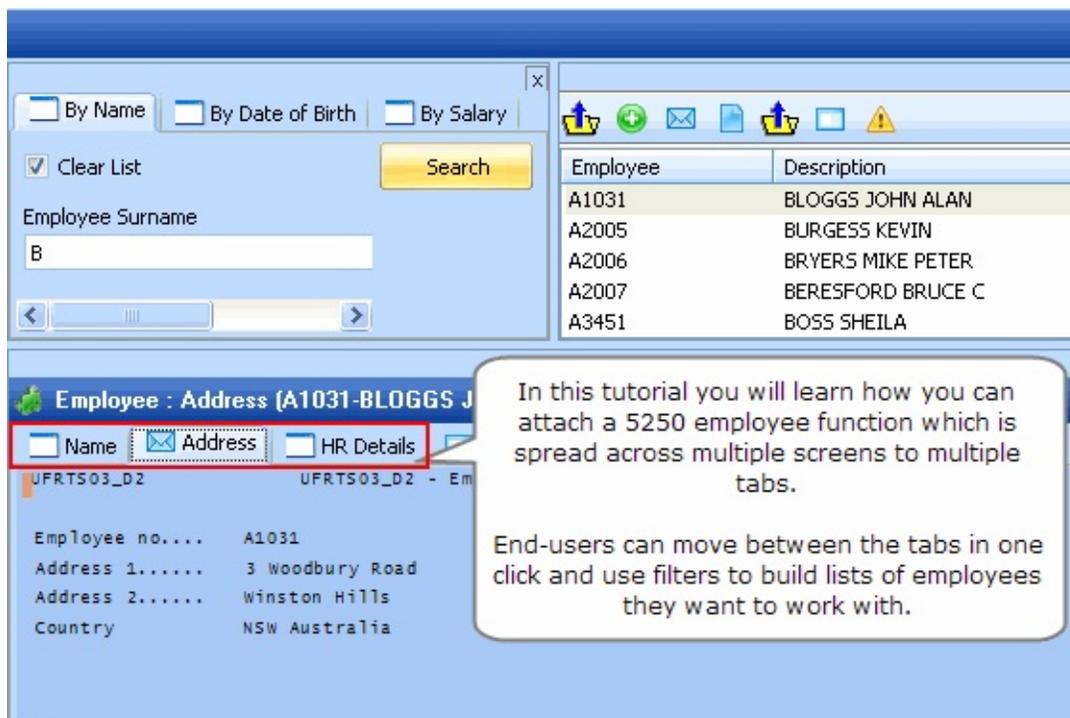
How to use and create SHARED properties and functions

RAMP-TSAD06: Handling Multiple Screens on Multiple Tabs

RAMP-TS 5250 destination screens are displayed on framework command handler tabs. Typically just one screen is displayed on a command handler tab. However it is possible to split multiple screens up across multiple tabs. This tutorial covers the concepts and skills required to do this.

Objectives

Learn how to attach a destination which has been spread across multiple screens to multiple tabs.



To achieve this objective, you will complete the following steps:

[RAMP-TSAD06 Step 1. A Multiple 5250 Screen Scenario](#)

[RAMP-TSAD06 Step 2. Name the Screens](#)

[RAMP-TSAD06 Step 3. Classify the Screens](#)

[RAMP-TSAD06 Step 4. Review and Understand the Targets List](#)

[RAMP-TSAD06 Step 5. Using Multiple Command Handler Tabs](#)

[RAMP-TSAD06 Step 6. Review and Alter Buttons and Function Keys](#)

[RAMP-TSAD06 Step 7. Review the value you have added to the 5250 application](#)

RAMP-TSAD06 Appendix: Function UFRTS03

Before You Begin

The following are assumed knowledge for the commencement of this tutorial.
You need to:

Understand the basic structure and mechanics of RAMP scripting.

Understand the concept of 5250 screens classified destinations, junctions or specials.

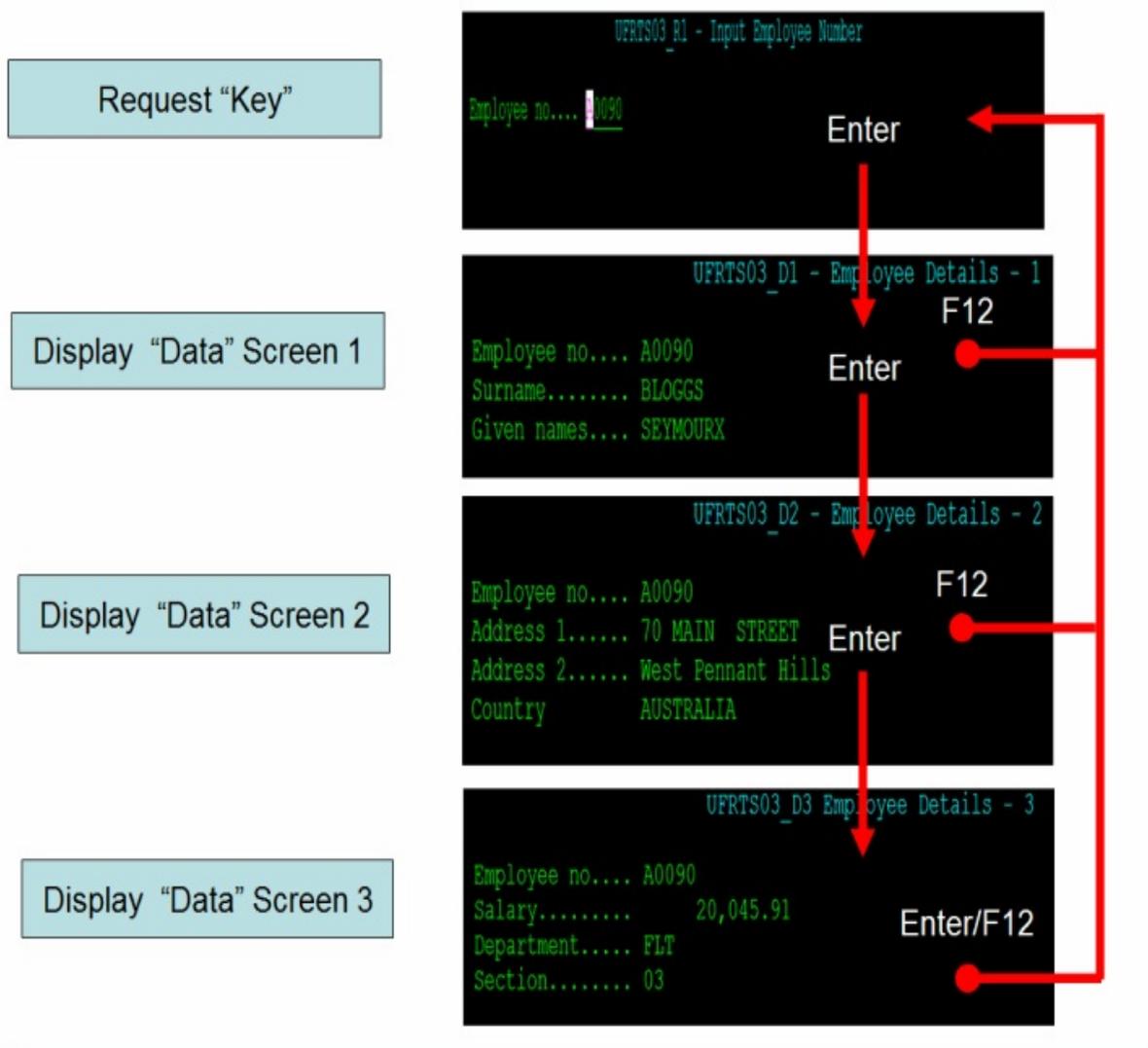
Understand how to snap a destination screen onto a framework command handler tab.

- If you attempt this tutorial without this assumed knowledge you will probably not be able to understand it.

RAMP-TSAD06 Step 1. A Multiple 5250 Screen Scenario

This tutorial uses a classic four screen 5250 Key -> Data inquiry sequence as its main scenario.

However, the Data part is too much to fit onto a single 5250 screen, so it has been spread across three 5250 screens like this:



These sample screens do not have much information on them. This is a deliberate choice to avoid blurring the objective of this tutorial.

You need to imagine that each of the 5250 "data" screens is packed full of information which is usually why it has been split across three 5250 screens.

Note how the function keys Enter and F12 have been used to perform classic

5250 style navigation. Also note that if you are on Data Screen 3, you cannot return to Data Screen 2 without going via the Request screen again. A minor design flaw which could possibly be quite irritating to people who use this 5250 application all the time.

If you are completing this tutorial as part of classroom training, the function for the displayed LANSAs 5250 screen will exist in your system in process UF_RTS, function UFRTS03.

If you are completing this tutorial as self-study, note that the source code is available in a function named UFRTS03 which can be found in [RAMP-TSAD06 Appendix: Function UFRTS03](#). It is recommended that you create a 5250 LANSAs process named UF_RTS, and in it a RDML function named UFRTS03 to which you copy the code. Then check the process and the function into your 5250 server so you can try out these scenarios in detail.

RAMP-TSAD06 Step 2. Name the Screens

In this step you will use RAMP Tools name the process menu and the four screens.

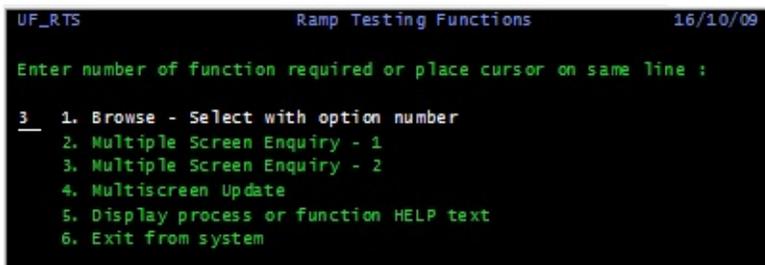
If you do not understand how to do this you should stop doing this tutorial and complete one of the core tutorials instead.

1. On the System i Main menu enter this command:

```
lansa run uf_rts partition(dem)
```

2. Name the screens:

Screen	Name You Should Use
RAMP Testing Functions menu	UF_RTS Also name the option field txtOption. If you are completing this tutorial in a classroom setting, you will need to select option 3 in the RAMP Testing Functions menu to bring up the correct screens:



```
UF_RTS Ramp Testing Functions 16/10/09
Enter number of function required or place cursor on same line :
3 1. Browse - Select with option number
  2. Multiple Screen Enquiry - 1
  3. Multiple Screen Enquiry - 2
  4. Multiscreen Update
  5. Display process or function HELP text
  6. Exit from system
```

Request "Key" Screen	UFRTS03_R1 Also name the employee number field used in navigation as txtEmpNo
Display "Data" Screen 1	UFRTS03_D1

Display UFRTS03_D2
"Data"
Screen 2

Display UFRTS03_D3
"Data"
Screen 3

RAMP-TSAD06 Step 3. Classify the Screens

1. After identifying all the required screens and naming the required fields on them, you should classify the screens.

Name You Used Classification of this Screen

UF_RTS	Junction
UFRTS03_R1	Junction
UFRTS03_D1	Destination
UFRTS03_D2	Destination
UFRTS03_D3	Destination

.

RAMP-TSAD06 Step 4. Review and Understand the Targets List

You should now have a junction screen (UFRTS03_R1) and three destination screens (UFRTS03_D1, UFRTS03_D2 and UFRTS03_D3) defined and scripted. Open RAMP Tools and click on each of the screen definitions to review their Targets lists.

Their respective Targets lists will look like this:

Screen Name	Type of Screen	Targets
UFRTS03_R1	Junction	UFRTS03_D1
UFRTS03_D1	Destination	UFRTS03_R1 UFRTS03_D2
UFRTS03_D2	Destination	UFRTS03_R1 UFRTS03_D3
UFRTS03_D3	Destination	UFRTS03_R1

If your target lists do not look like this, you should continue to manually demonstrate screen navigations via RAMP Tools until they do.

The Targets associated with a screen definition are very important. They tell the RAMP navigation planner what screens a particular screen can navigate to (ie: target). Normally the Targets are automatically updated when you manually demonstrate a navigation to the RAMP Tools editor.

As you become more experienced with RAMP you may decide to just update the Targets list manually and add the appropriate code to the screen's navigation handler function. In effect this is exactly what demonstrating a navigation via RAMP Tools does.

The Targets lists used here are simple to understand:

The junction UFRTS03_R1's target list effectively says "My vHandle_NAVIGATETO function contains scripts that can navigate to destination UFRTS03_D1".

The destination UFRTS03_D1's target list effectively says "My vHandle_NAVIGATE function contains scripts that can navigate to junction UFRTS03_R1 or to destination UFRTS03_D2".

The presence of junction UFRTS03_R1 in the target lists of all 3 destinations is significant. The first junction in a destination's target list is called the exit junction. The exit junction is used to get out of the destination and back on to the junction "freeway or motorway" (ie: the network of identified junctions that are used to rapidly move between destinations). If a destination screen is on display, it will be asked to navigate to its nominated "exit junction" before a navigation to another destination (including itself).

RAMP-TSAD06 Step 5. Using Multiple Command Handler Tabs

In handling this type of scenario the first option you have available is to put each of the destination screens onto its own command handler tab.

This adds value to the 5250 application in which the user cannot immediately go to data screen 3 (say) without having to go through data screens 1 and 2. Now they can move freely among all 3 data screens without having to cancel and go back to the key screen.

1. In the Framework, add three new instance level commands to the Employee business object: Name, Address and HR Details.
2. Make Name the default command and set the sequence of the commands to 1, 2 and 3. Resequence the other commands associated with Employee to that they come after these three commands.
3. In RAMP Tools link the destination screens with the commands:

UFRTS03_D1 Name

UFRTS03_D2 Address

UFRTS03_D3 HR Details

4. Modify the script of the UFRTS03_R1. You need to change the SETVALUE so that the commands will be shown for the employee selected in the instance list.

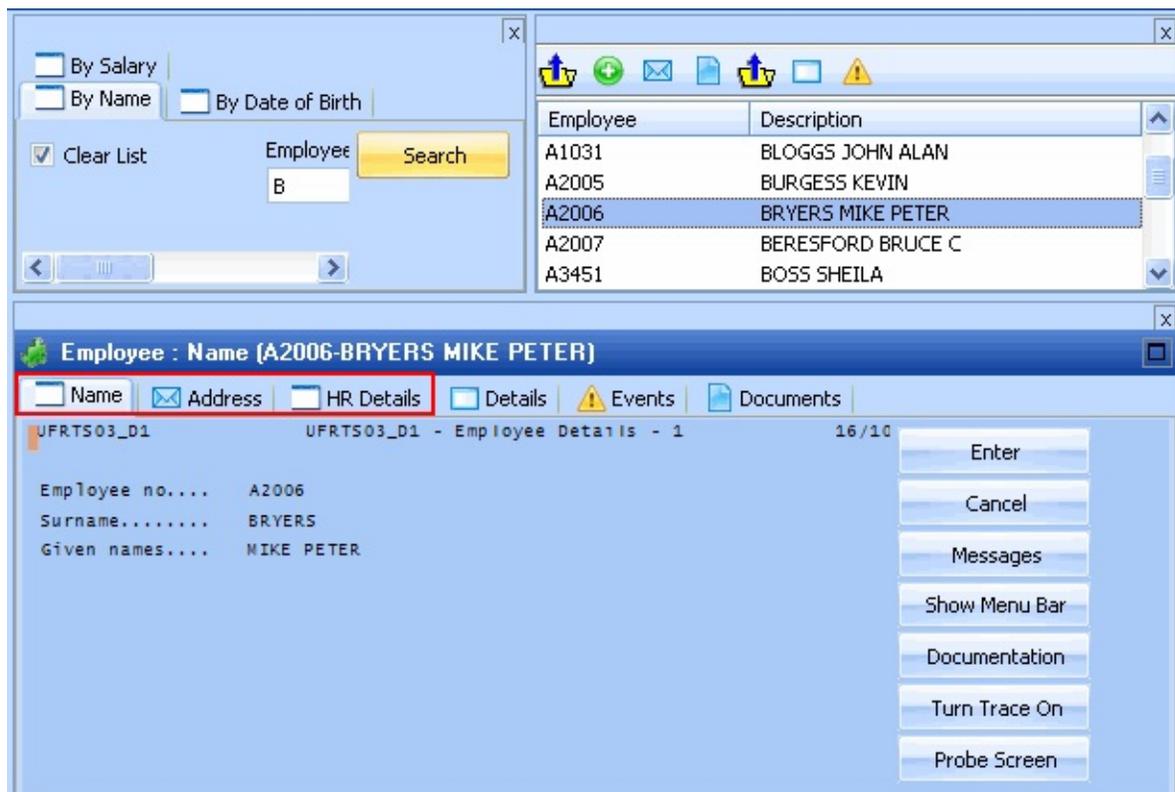
```
case "UFRTS03_D1":  
{  
  
    /* Set up data fields on form UFRTS03_R1 */  
  
    SETVALUE("EmpNo",objListManager.AKey1[0]);  
    SENDKEY(KeyEnter);  
  
    /* Check for arrival at UFRTS03_D1 */
```

```
Q_CHECK_CURRENT_FORM("UFRTS03_D1","Unable to navigate
to form UFRTS03_D1");
}
break;
```

You may want to review [RAMP-TS006 Step 2. Change the Script to Use the Current Instance List Entry](#).

5. Save and restart the Framework.

When you display an employee the command tabs should now look like this:



Notice that you can display the three 5250 screens in any combination.

You may have to trace and modify your scripts, or even demonstrate new navigations to get this example to function correctly. This is a normal part of scripting 5250 screen interactions. You should persist with doing this until all three screens function correctly. Previously completed tutorials should have

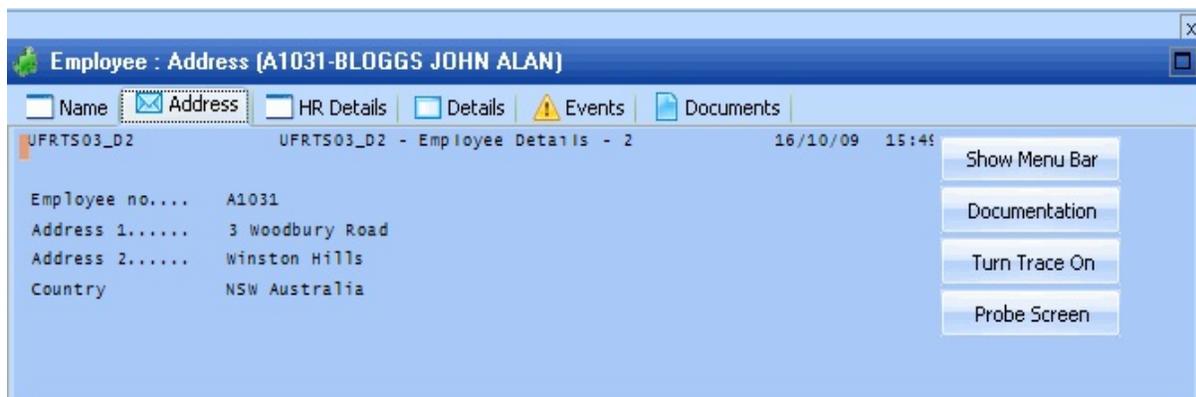
equipped you with the skills required to debug your scripts until they function correctly.

RAMP-TSAD06 Step 6. Review and Alter Buttons and Function Keys

All the 5250 function keys are related to navigation activities and you have replaced all the navigation with something better. The 5250 Enter to go forward, and F12 to Cancel, operations are largely nonsensical in a windows application.

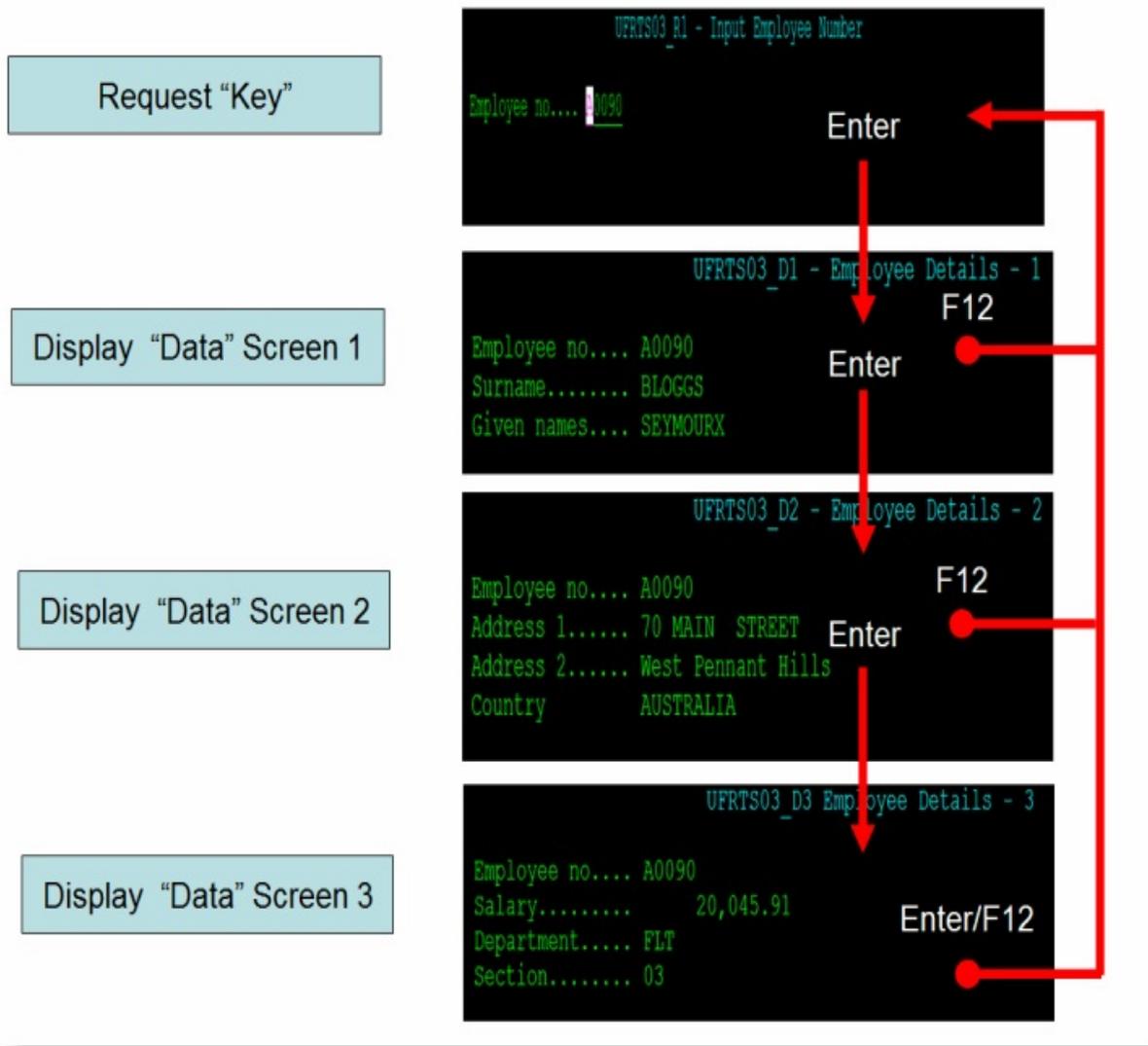
For example: You do not "cancel" an inquiry about the Address of employee A0090, you simply move on to what you want to do next – display the Name details of employee A0070 (say). You should strive to achieve this in your modernization project because it makes your application more familiar and comfortable to people who are used to the Windows interface.

1. In RAMP-Tools change destination screens UFRTS03_D1, UFRTS03_D2 and UFRTS03_D3 so that no function keys or buttons are enabled.
2. Do a partial save.
3. In the Framework, display another employee. Only the design-time buttons are now shown:

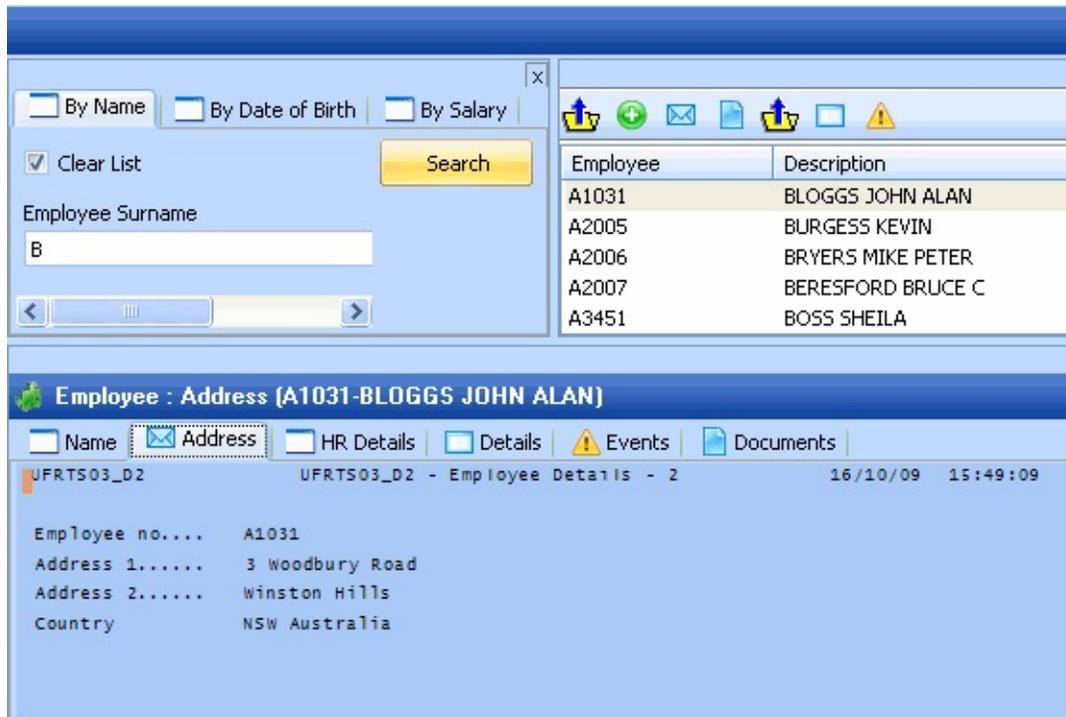


RAMP-TSAD06 Step 7. Review the value you have added to the 5250 application

You started with a very simple 5250 four screen inquiry that could only do this:



Now it looks like this:



The user can now build lists of the employees they want to work with.

They can move from displaying the Name details of employee A0070, to their HR details in one click, and then go back again in one click.

They can move from displaying the Name details of employee A0070 to the HR Details of employee A1031 in two clicks.

RAMP-TSAD06 Appendix: Function UFRTS03

```
FUNCTION OPTIONS(*DIRECT)
```

```
BEGIN_LOOP
```

- * Get the key details
- * A classic junction
- * Cancel key goes back to process menu or caller program

```
RQ1: DOUNTIL COND('#IO$STS = OK')
```

```
REQUEST FIELDS(#EMPNO) EXIT_KEY(*NO) PANEL_ID(UFRTS03_R1)  
FETCH FIELDS(*ALL) FROM_FILE(pslmst) WITH_KEY(#EMPNO) ISSU  
ENDUNTIL
```

- * Display details screen 1
- * Cancel key goes back to request next employee number
- * Enter goes forward to Details screen 2

```
DISPLAY FIELDS(#EMPNO #SURNAME #GIVENAME) EXIT_KEY(*NO
```

- * Display details screen 2
- * Cancel key goes back to request next employee number
- * Enter goes forward to Details screen 3

```
DISPLAY FIELDS(#EMPNO #ADDRESS1 #ADDRESS2 #ADDRESS3) EX
```

- * Display details screen 3
- * Cancel key goes back to request next employee number
- * Enter goes forward to request next employee number

```
DISPLAY FIELDS(#EMPNO #SALARY #DEPARTMENT #SECTION) EXIT_I
```

- * Loop around and ask for the next employee number

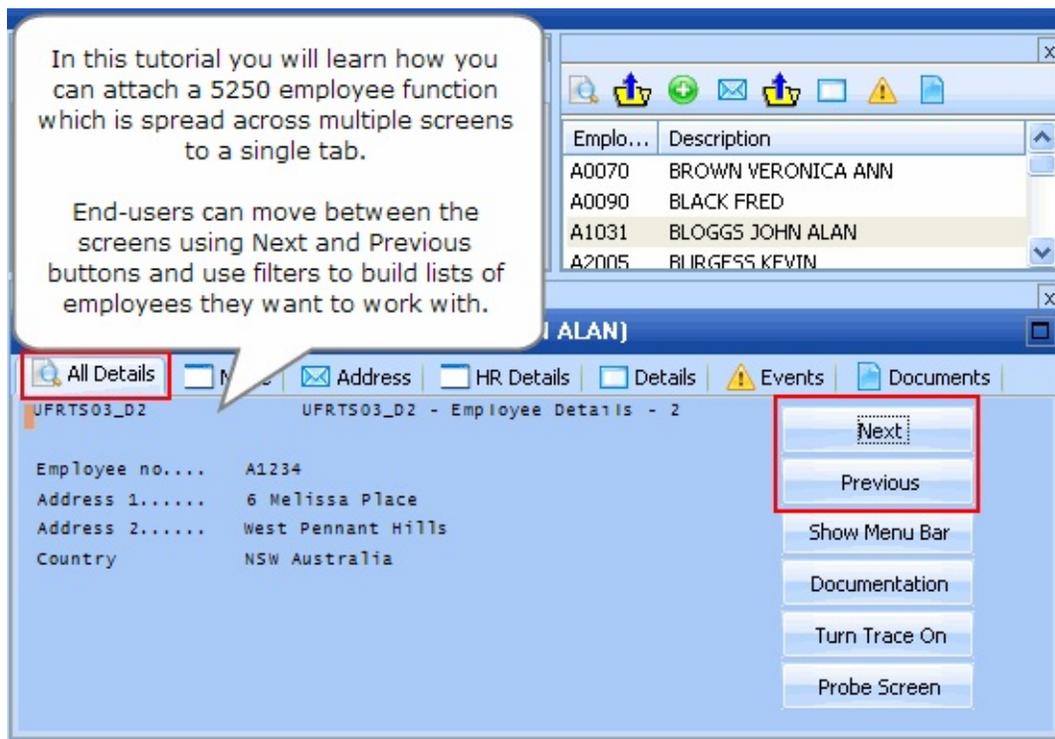
```
END_LOOP
```

RAMP-TSAD07: Handling Multiple Screens on a Single Tab

RAMP-TS 5250 destination screens are displayed on framework command handler tabs. Typically just one screen is displayed on a command handler tab. However it is possible to display multiple 5250 screens on a single tab. This tutorial covers the concepts and skills required to do this.

Objectives

Learn how to attach a destination which has been spread across multiple screens to a single tab and how enable buttons to navigate between the screens.



-
- To achieve this objective, you will complete the following steps:
- [RAMP-TSAD07 Step 1. A Multiple 5250 Screen Scenario](#)
- [RAMP-TSAD07 Step 2. Making a Plan](#)
- [RAMP-TSAD07 Step 3. Putting the Screens on a Single Tab](#)
- [RAMP-TSAD07 Step 4. Enable Function Keys/Buttons and Add Required Scripting](#)
- [RAMP-TSAD07 Step 5. Defining the Exit Junctions and vHandle_NAVIGATETO scripting](#)

- [RAMP-TSAD07 Step 6. Testing and Debugging](#)
- [RAMP-TSAD07 Step 7. Fine Tuning](#)

Before You Begin

The following are assumed knowledge for the commencement of this tutorial.
You need to:

Have completed the preceding mini-tutorial [RAMP-TSAD06: Handling Multiple Screens on Multiple Tabs](#) . If you attempt this tutorial without this assumed knowledge you will probably not be able to understand it.

RAMP-TSAD07 Step 1. A Multiple 5250 Screen Scenario

This tutorial continues to use the same classic four screen 5250 Key -> Data inquiry function UFRTS03 as its scenario as the previous tutorial. See [RAMP-TSAD06 Step 1. A Multiple 5250 Screen Scenario](#) .

RAMP-TSAD07 Step 2. Making a Plan

In the [RAMP-TSAD06: Handling Multiple Screens on Multiple Tabs](#) tutorial these screens were identified and made to function on three command handler tabs.

Now we are going to make all three destination screens appear on a single command tab named All Details and allow the user to move between them with Previous and Next buttons. To do this, we need to make a plan, something like this:

Screen	Associated with Command Tab	Enabled Buttons / Function Keys	Targets
UFRTS03_D1	All Details	Next/Enter -> UFRTS03_D2	UFRTS03_R1
UFRTS03_D2	-	Previous/F12 - > UFRTS03_D1 Next/Enter -> UFRTS03_D3	UFRTS03_R1
UFRTS03_D3	-	Previous/F12 -> UFRTS03_D2	UFRTS03_R1

Looking at this plan in more detail, you should be able to answer these questions:

Question	Notes
Why is only UFRTS03_D1 linked to a command handler tab?	There is going to only be one command now, All Details and when the user executes it the screen UFRTS03_D1 will be displayed on a command tab.

<p>Why are UFRTS03_D2 and UFRTS03_D2 not linked to any command handler tabs.</p>	<p>They are not linked to any commands themselves. The only way to get to screens UFRTS03_D2 or UFRTS03_D3 is to go via UFRTS03_D1, then used the Next button(s) to advance to them. If you want them to be independently and directly accessible put them on their own command tabs.</p>
<p>When screen UFRTS03_D1 is displayed what function keys / buttons will be enabled and what will they do?</p>	<p>It will have a Next button and the Enter key enabled. When used they will cause screen UFRTS03_D2 to display.</p>
<p>When screen UFRTS03_D2 is displayed what function keys / buttons will be enabled and what will they do?</p>	<p>It will have a Previous button and the F12 key enabled. When used they will cause screen UFRTS03_D1 to (re)display. It will have a Next button and the Enter key enabled. When used they will cause screen UFRTS03_D3 to display.</p>
<p>When screen UFRTS03_D3 is displayed what function keys / buttons will be enabled and what will they do?</p>	<p>It will have a Previous button and the F12 key enabled. When used they will cause screen UFRTS03_D2 to (re)display.</p>
<p>Why do all 3 destination screens have an exit</p>	<p>Any displayed destination needs to have an exit junction specified and the appropriate code in its vHandle_NAVIGATETO script to navigate to that junction.</p>

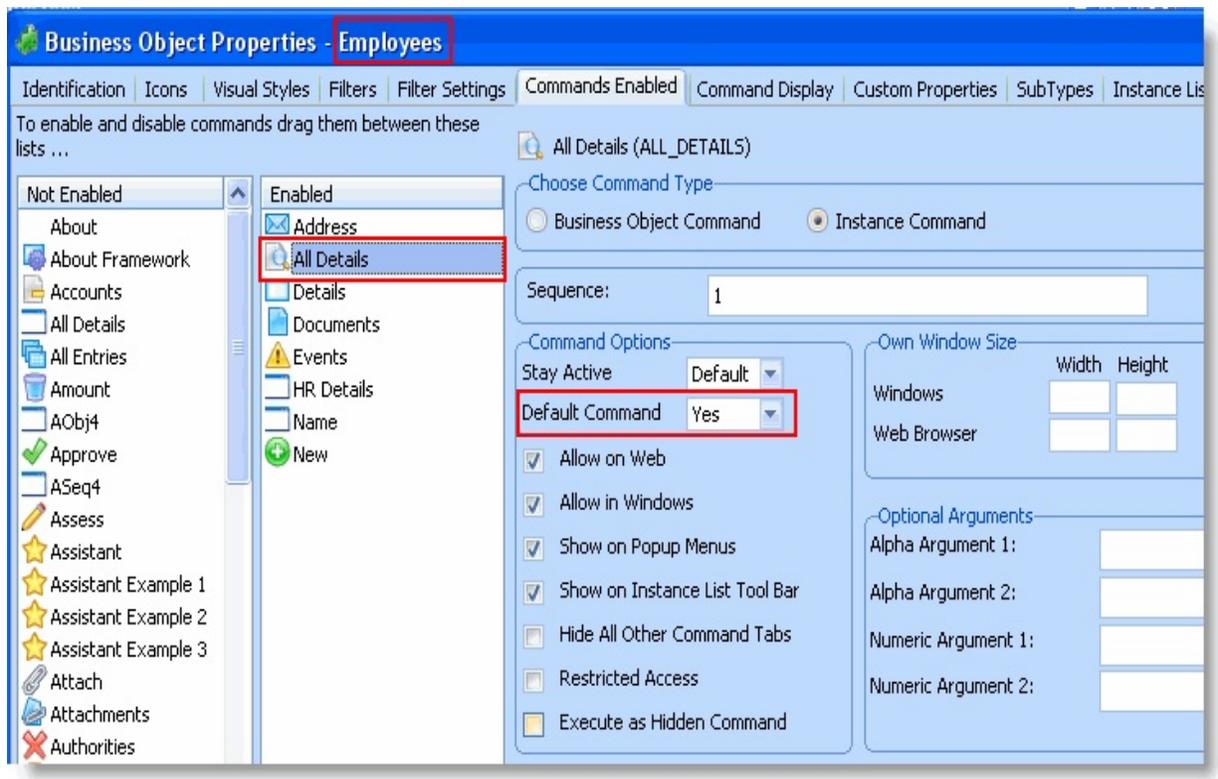
junction?	
How is the exit junction used?	When a destination screen is displayed it can at any time be replaced by another destination, possibly for another business object or business object instance (ie: another order, product or customer, say). To allow this to happen it should have an exit junction that allows the RAMP navigator to exit from it and get back onto the junction freeway/motorway and plan the fastest route to the next destination.

The next few steps in this tutorial will cover implementing this plan and testing/debugging it.

RAMP-TSAD07 Step 3. Putting the Screens on a Single Tab

In this step you will attach the screens to a single command handler tab.

1. In the Framework, associate the All Details command with the Employee business object.
2. Make it the Default command.



3. In RAMP Tools locate the details of the UFRTS03_D1 screen and remove the association with the Name command, then associate the screen with the All Details command.
4. Remove the command handler associations from UFRTS03_D2 and UFRTS03_D3.
5. Do a partial save of the RAMP definition.
6. Save and restart the Framework.

You now have a command handler tab captioned All Details which you can use to display the basic details (screen UFRTS03_D1) for any selected employee:

By Name
By Date of Birth
By Salary
x

Clear List
 Search

Employee Surname

<
>

Employee	Description
A0070	BROWN VERONICA ANN
A0090	BLACK FRED
A1031	BLOGGS JOHN ALAN
A2005	BURGESS KEVIN
A2006	BRYERS MIKE PETER
A2007	BERESFORD BRUCE C
A3451	BOSS SHEILA
A3487	BENTLEY BRIDGET
A3564	BROWN FREDDY
A4567	BLACK MAY
A6663	BLACKMORE RICHIE

Employee : All Details (A2005-BURGESS KEVIN)

All Details
Name
Address
HR Details
Details
Events
Documents

UFRTS03_D1
UFRTS03_D1 - Employee Details - 1
20/10/09 12:51:21

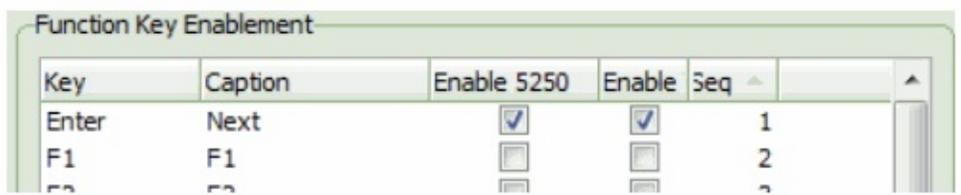
Employee no.... A1234
 Surname..... JACKSON
 Given names.... STEPHEN

RAMP-TSAD07 Step 4. Enable Function Keys/Buttons and Add Required Scripting

In this step you will enable Next and Previous buttons on the screens to allow the end-user to navigate between the screens in the function. You will also make some changes to the scripts.

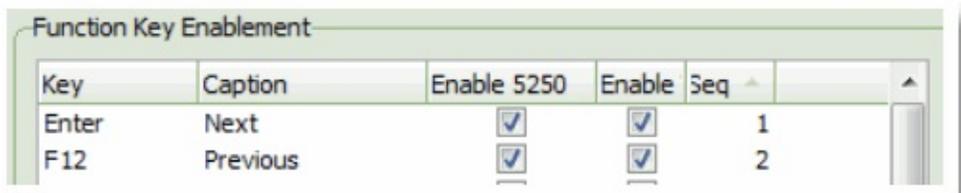
1. Review the properties of screens UFRTS03_D1, UFRTS03_D2 and UFRTS03_D3 and enable buttons/function keys as previously planned.

UFRTS03_D1 should look like this:



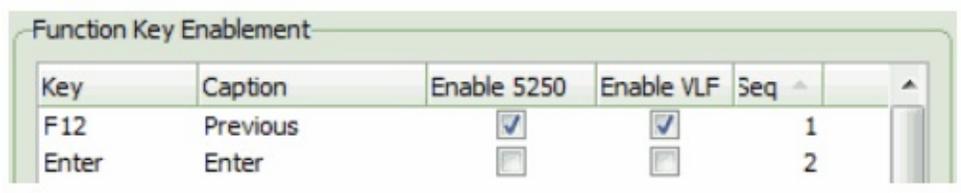
Key	Caption	Enable 5250	Enable	Seq	
Enter	Next	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	
F1	F1	<input type="checkbox"/>	<input type="checkbox"/>	2	

UFRTS03_D2 should look like this:



Key	Caption	Enable 5250	Enable	Seq	
Enter	Next	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	
F12	Previous	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	2	

UFRTS03_D3 should look like this:



Key	Caption	Enable 5250	Enable VLF	Seq	
F12	Previous	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	
Enter	Enter	<input type="checkbox"/>	<input type="checkbox"/>	2	

2. Save your changes and restart the Framework.

Next, review and alter the vHandle_BUTTONCLICK scripts of each of these screens to correctly handle the button navigations.

3. First, refer to the 5250 navigation picture in [RAMP-TSAD06 Step 1. A Multiple 5250 Screen Scenario](#).
4. From this navigation picture you can see pretty easily what needs to be done:

Screen	Button/Function Key	What screen should result?	What vHandle_BUTTONCLICK needs to do
UFRTS03_D1	Next/Enter	UFRTS03_D2	Send enter key. The default button script should handle this.
UFRTS03_D2	Next/Enter	UFRTS03_D3	Send enter key. The default button script should handle this.
UFRTS03_D2 (see note)	Previous/F12	UFRTS03_D1	Send F12 to get to UFRTS03_R1 (junction). Send Enter to advance to UFRTS03_D1. The default button script will not handle this.
UFRTS03_D3 (see note)	Previous/F12	UFRTS03_D2	Send F12 to get to UFRTS03_R1 (junction). Send Enter to advance to UFRTS03_D1. Send Enter to advance to UFRTS03_D2. The default button script will not handle this.

Note the addition of the Previous/F12 operations to UFRTS03_D2 and to UFRTS03_D3. In the underlying 5250 application no such direct navigations exist (ie: you cannot actually go from UFRTS03_D2 to UFRTS03_D1 in one operation).

However, with some simple scripting you can make it appear to the user as if this feature actually exists. This is another simple example of adding value to an existing 5250 application.

5. Change your vHandle_BUTTONCLICK functions. No changes is required to the UFRTS03_D1 script.

In **UFRTS03_D2** use a button click switch construct like this:

```
switch (sButton)
{
    case KeyEnter: /* Enter-
Next means move forward to UFRTSD03_03 */
        SENDKEY(KeyEnter);
        break;
    case KeyF12: /* F12-Previous means go back to UFRTSD03_01 */
        Q_SENDKEY("",KeyEnter);
        SENDKEY(KeyF12);
        break;
    default:
        ALERT_MESSAGE("Invalid function key used");
        break;
}
```

In **UFRTS03_D3** use a button click switch construct like this:

```
switch (sButton)
{
    case KeyF12: /* F12-Previous means go back to UFRTSD03_02 */
        Q_SENDKEY("",KeyEnter);
        Q_SENDKEY("",KeyEnter);
        SENDKEY(KeyF12);
        break;
    default:
        ALERT_MESSAGE("Invalid function key used");
        break;
}
```

Note:

These RAMP scripts have been changed to reinterpret what using F12 actually means. On a RAMP tab showing the 5250 screen UFRTS03_D3, the F12 function key now means send F12, Enter, Enter to the 5250 server. This

reinterpretation has added value to the underpinning 5250 application because it has enabled the user to move directly to the previous screen from this screen. A lot of quite valuable business process improvement is underpinned by very simple strategies like this.

All invalid function keys now result in an alert message. This is a fail safe only. RAMP should prevent the keys from being used anyway.

The use of the Q_SENDKEY() functions. RAMP-TS SENDKEY() operations are asynchronous, so only the first SENDKEY() request can be sent immediately. The subsequent requests need to be queued up and handled when the resulting screen(s) arrive back asynchronously.

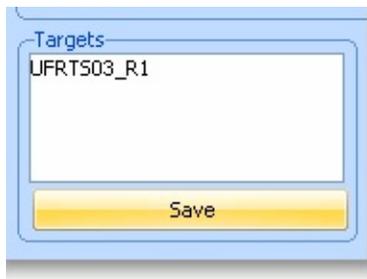
7. Save your script changes and do a partial save of the RAMP definition.

Don't test your changes until you complete the next step. Without them your application's navigation may become "stuck", requiring you to cancel the application.

RAMP-TSAD07 Step 5. Defining the Exit Junctions and vHandle_NAVIGATETO scripting

The final thing you need to do is make sure that all three destinations have an Exit Junction specified and that their vHandle_NAVIGATETO functions can navigate to the exit junction when requested.

1. Open RAMP Tools and review the properties of the 3 destination screens UFRTS03_D1, UFRTS03_D2 and UFRTS03_D3.
2. In each destination screen, edit the Targets list displayed so that it only contains the name of the junction UFRTS03_R1. Remember to click the save button after making your changes to each Targets list.



Technical Note about Exit Junctions: When a destination screen is on display and a navigation to another screen needs to be performed, the destination screen will be asked to navigate to its exit junction before the navigation route to the target screen is calculated. By default a destination screen's exit junction is the first junction defined in its Targets list.

3. Now review the script associated with each of the destination screen.
4. Locate its vHandle_NAVIGATETO function and ensure it contains a script that can handle a request to navigate to the junction named UFRTS03_R1.

Your scripts probably contain this code already from the previous tutorial. However, it's important you understand that you can (and sometimes do) manually edit the Targets list and add the associated implementation logic to the vHandle_NAVIGATETO function.

In their trimmest form, the vHandle_NAVIGATETO functions for all of the destination screens should now look like this:

```
switch (sToForm)
```

```
{  
  
    /* <NAVIGATESWITCH> - Do not remove or alter this line */  
  
    case "UFRTS03_R1": /* Navigate back to the exit junction */  
    {  
        SENDKEY(KeyF12);  
        Q_CHECK_CURRENT_FORM("UFRTS03_R1","Unable to navigate t  
    }  
    break;  
  
    default: /* Handle an invalid request */  
        alert("Form " + this.vName + " cannot navigate to form " + sToForm);  
        bReturn = false;  
        break;  
  
}
```

RAMP-TSAD07 Step 6. Testing and Debugging

The final step involves some extensive testing of the navigation defined in your code.

Check that your All Details command handler tab can handle these situations.

Repeats the Next->Next->Previous cycle around the 3 different destination screens in different variations and combinations.

Selecting different employees down the instance list with UFRTS03_D1 on display.

The Employee Details 1 screen (UFRTS03_D1) returns to its nearest junction, Input Employee Number screen (UFRTS03_R1). The NAVIGATETO script for UFRTS03_R1 navigates back to UFRTS03_D1 by setting Employee Number from the instance list and sending the Enter key.

Selecting different employees down the instance list with UFRTS03_D2 displayed initially.

The Employee Details 1 screen (UFRTS03_D1) is displayed for each employee selected in the instance list. This happens because screen UFRTS03_D1 is associated with command handler “All Details”.

Selecting different employees down the instance list with UFRTS03_D3 displayed initially.

This action will once again displayed the first Employee Details screen (UFRTS03_D1) for the employee selected from the instance list.

Navigating to other business objects and back again with UFRTS03_D1 displayed.

When you select a different business object, RAMP exits the current screen to return to its nearest junction. From here, RAMP creates a navigation plan to reach the requested command handler for the currently selected business object. i.e. a different 5250 screen.

When you reselect the All Details tab, the opposite navigation occurs. The current screen exits to its nearest junction and a RAMP generated navigation plan returns to the first All Details screen (UFRTS03_D1).

Navigating to other business objects and back again with UFRTS03_D2 displayed (should come back to UFRTS03_D1)

The navigation is similar to the previous example. Once again when you reselect the All Details tab for the Employees business object it is the first Employee Details screen (UFRTS03_D1) which is displayed since it is this screen which is associated with All Details.

Navigating to other business objects and back again with UFRTS03_D3 displayed (should come back to UFRTS03_D1).

The logic here is the same as the previous example.

If you have problems you should use application level tracing to see what is happening as your scripts execute.

RAMP-TSAD07 Step 7. Fine Tuning

While testing your Next -> Next -> Previous button processing you may have noticed a flicker as RAMP-TS navigates through intermediate screens.

This may be particularly noticeable when using the Previous button on screen UFRTS03_D3, because the underlying 5250 application design means that you need to do this:

```
switch (sButton)
{
  case KeyF12: /* F12-Previous means go back to UFRTSD03_02 */
    Q_SENDKEY("",KeyEnter);
    Q_SENDKEY("",KeyEnter);
    SENDKEY(KeyF12);
    break;
  default:
    ALERT_MESSAGE("Invalid function key used");
    break;
}
```

The flicker is because you started this button handling sequence with the current 5250 screen visible.

Your script is not quite complete yet and needs to be fined tuned. Change it to this:

```
switch (sButton)
{
  case KeyF12: /* F12-Previous means go back to UFRTSD03_02 */
    HIDE_CURRENT_FORM();
    Q_SENDKEY("",KeyEnter);
    Q_SENDKEY("",KeyEnter);
    SENDKEY(KeyF12);
    break;
  default:
    ALERT_MESSAGE("Invalid function key used");
    break;
}
```

The HIDE_CURRENT_FORM() function call causes the current RAMP-TS

screen to be hidden, preventing the intermediate screen navigations from being seen (ie: the flashes).

The RAMP_TS screen becomes visible again when the SHOW_FORM function in the vHandle_ARRIVE script of the ultimate destination is executed.

RAMP-TSAD08: Screen Wrapper Basics

This tutorial shows the basic steps in creating a screen wrapper.

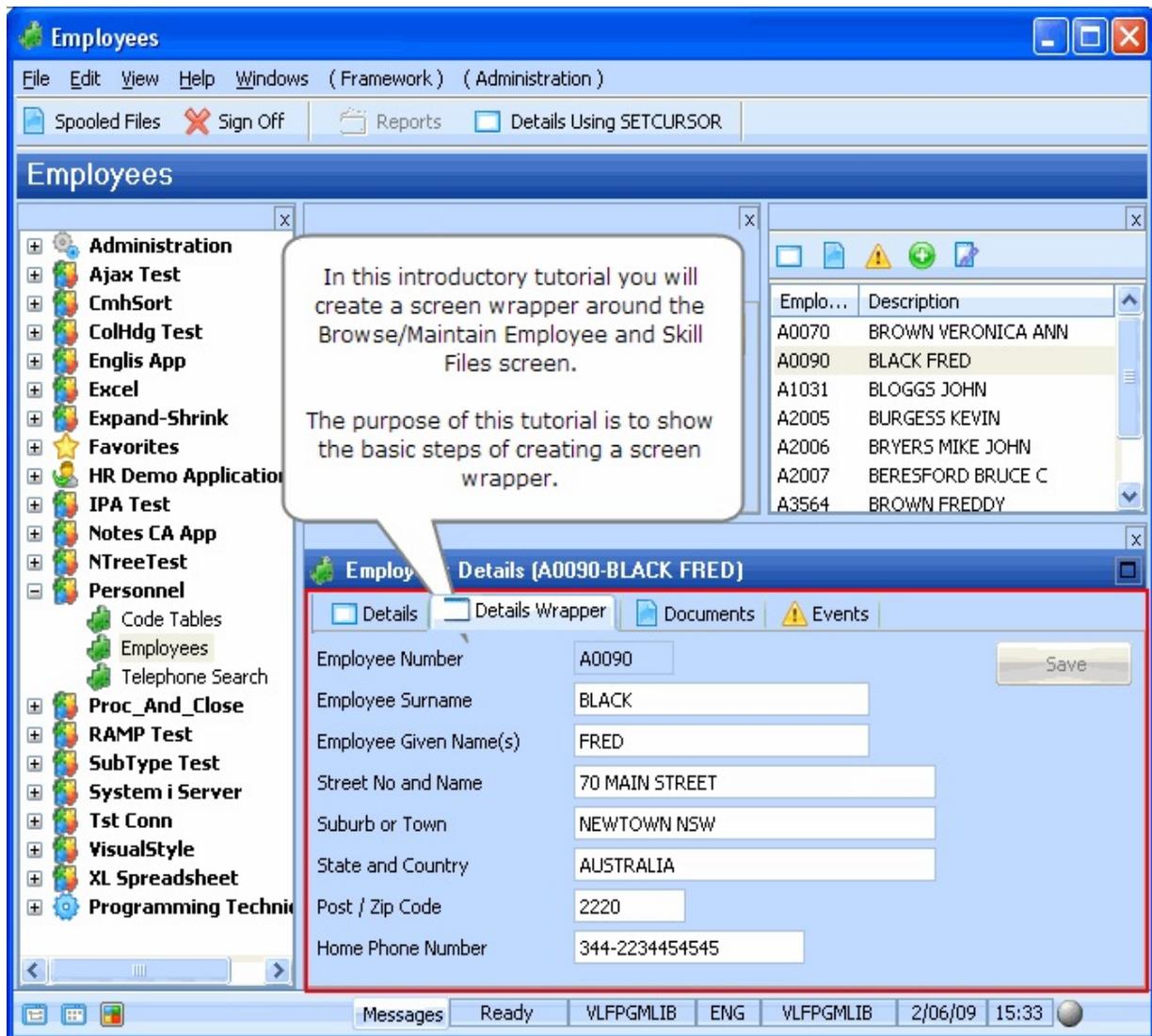
A screen wrapper is a Visual LANSA reusable part which can navigate to one or more 5250 screens and get and set values on the screens. The wrappers offer a way to enhance the existing 5250 application without having to rewrite it.

The screen wrapper becomes the user interface, but underneath the existing 5250 application is still being used.

This is simply an introductory exercise. There would be no real reason to create a screen wrapper over this 5250 screen.

Objectives

Understand the basics of creating screen wrappers



To achieve this objective, you will complete the following steps:

- [RAMP-TSAD08 Step 1. Name the Fields to Be Used in the Wrapper](#)
- [RAMP-TSAD08 Step 2. Create and Snap in the Screen Wrapper](#)
- [RAMP-TSAD08 Step 3. Understanding the Screen Wrapper Code](#)
- [RAMP-TSAD08 Step 4. Test Updating the Screen from the Wrapper](#)
- [Summary](#)

Before You Begin

In order to complete this tutorial, you must have completed the core tutorials RAMP-TS001 - RAMP-TS015.

RAMP-TSAD08 Step 1. Name the Fields to Be Used in the Wrapper

In this step you will name all the fields on the DisplayEmployee screen so that they can be used in a screen wrapper.

1. Start RAMP Tools.
2. Start the RAMP-TS 5250 emulator.
3. Navigate to the Browse/Maintain Employee and Skill Files screen.

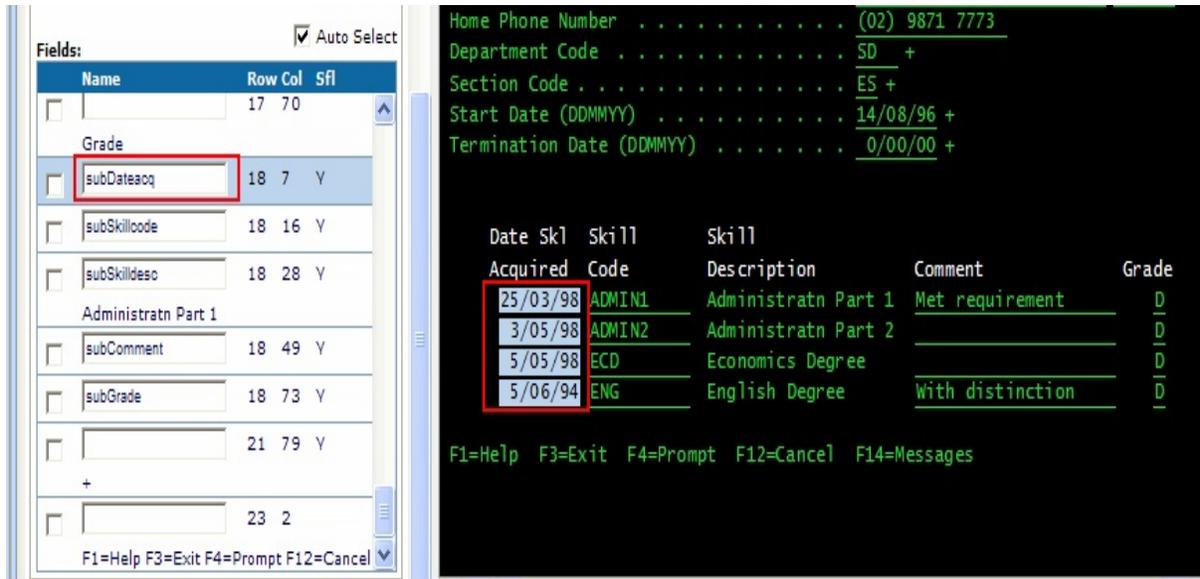
```
LANSA run pslsys partition(dem)
```

4. Select option 3 in the Personnel System main menu.
5. Specify an employee number, for example A1234.
6. Press F21 to put the screen in input mode.
7. Name the fields on the screen (you have named some of them in the previous tutorials). The fields should be named like this:

Employee Number	txtEmpno
Employee Surname	txtSurname
Employee Given Name(s)	txtGivenname
Street No and Name	txtAddress1
Suburb or Town	txtAddress2
State and Country	txtAddress3
Post Code	txtPostcode
Home Phone Number	txtPhone
Department Code	utxtDepartment
Section Code	SECTION
Start Date	Date_Start_DDMMYY
Termination Date	Date_Termn_DDMMYY

You need to also name the subfile columns in the skills table:

8. Select the Date Skl Acquired column (not the column heading) and name it subDateacq.



9. Name the rest of the columns like this:

Skill Code	subSkillcode
Skill Description	subSkilldesc
Comment	subComment
Grade	subGrade
the plus indicator	moreindicator

Note that you will not use all the fields you have named in the first part of the tutorial.

10. Save the screen definition.

Now that you have named all the fields on the screen, you are now ready to put it inside a screen wrapper.

RAMP-TSAD08 Step 2. Create and Snap in the Screen Wrapper

In this step you will copy the code for your first screen wrapper and then snap the screen wrapper to the Framework.

1. In the Visual LANSa editor, create a new reusable part with the name iiiRMP03. Make the description DisplayEmployee Wrapper.
2. Replace the code in the reusable part with this code:

```
Function Options(*DIRECT)
BEGIN_COM ROLE(*EXTENDS #VF_AC010) HEIGHT(227)
LAYOUTMANAGER(#MAIN_LAYOUT) WIDTH(497)
*
=====
* Simple Field and Group Definitions
*
=====
Group_By Name(#XG_HEAD) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #PHONEHME
#DEPARTMENT #SECTION)
* Body and Button arrangement panels
DEFINE_COM CLASS(#PRIM_PANL) NAME(#BUTTON_PANEL)
DISPLAYPOSITION(2) HEIGHT(227) HINT(*MTXTDF_DET1)
LAYOUTMANAGER(#BUTTON_FLOW) LEFT(409)
PARENT(#COM_OWNER) TABPOSITION(3) TABSTOP(False) TOP(0)
WIDTH(88)
DEFINE_COM CLASS(#PRIM_PANL) NAME(#BODY_HEAD)
DISPLAYPOSITION(1) HEIGHT(227) HINT(*MTXTDF_DET1)
LAYOUTMANAGER(#BODY_HEAD_FLOW) LEFT(0)
PARENT(#COM_OWNER) TABPOSITION(2) TABSTOP(False) TOP(0)
VERTICALSCROLL(True) WIDTH(409)
* 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)
READONLY(True) 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)
* 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)
DISPLAYPOSITION(3) HEIGHT(227) PARENT(#COM_OWNER)
WIDTH(409)
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
* In the command's uInitialize method routine, set the screen wrapper's
uCommand property to #com_owner
Set Com(#myscreen_wrapper) Ucommand(#com_owner)
Endroutine
* -----
-----
* Handle Command Execution
*
* You may also disable the entire form to prevent any input while RAMP is
navigating
* -----
-----
Mthroutine Name(uExecute) Options(*REDEFINE)
Invoke Method(#Com_Ancesor.uExecute)
#myscreen_wrapper.makerampTSAvailable
Set Com(#Save_Button) Enabled(False)
#com_owner.enabled := false
Endroutine
*
=====
* Event Handlers
*
=====
* RAMP has signalled it's ready. Invoke your navigation here.
* Once the navigaton starts, processing resumes in the vHandleArrive event
handler.
Evtroutine Handling(#myscreen_wrapper.RampTSAvailable)
Invoke Method(#myscreen_wrapper.navigatetoscreen)
Name('DisplayEmployee')
Endroutine

Evtroutine Handling(#myscreen_wrapper.vHandleArrive)
Arrivedscreen(#CurrentScreen) Previouscreen(#PreviousScreen)
Arrivedpayload(#Payload)

```

```

Case (#CurrentScreen)
When Value_Is(= 'DisplayEmployee')
Set Com(#SAVE_BUTTON) Enabled(false)
* Error handling: Payloads are destroyed when the ARRIVE script finishes
executing. Therefore, a payload of UPDATE_EMPLOYEE would most likely
mean there was a validation error.
If (#Payload = UPDATE_EMPLOYEE)
Use Builtin(message_box_show) With_Args(ok ok info "Validation Error"
"Please correct any errors")
Else
* Unlock the framework
Set Com(#avFrameworkManager) Ulocked(false)
#myscreen_wrapper.getvalue From('txtEmpno') Value(#empno.value)
#myscreen_wrapper.getvalue From('txtSurname') Value(#surname.value)
#myscreen_wrapper.getvalue From('txtGivename') Value(#givename.value)
#myscreen_wrapper.getvalue From('txtAddress1') Value(#address1.value)
#myscreen_wrapper.getvalue From('txtAddress2') Value(#address2.value)
#myscreen_wrapper.getvalue From('txtAddress3') Value(#address3.value)
#myscreen_wrapper.getvalue From('txtPhone') Value(#phonehme.value)
#myscreen_wrapper.getvalue From('txtPostcode') Value(#POSTCODE.value)

#com_owner.enabled := true
Endif
Endcase
Endroutine
* -----
* Handle Save
* -----
Mthroutine Name(Save)
* Set the 5250 field values to the values from this panel
#myscreen_wrapper.setvalue Infield('txtSurname') Value(#surname.value)
#myscreen_wrapper.setvalue Infield('txtGivename') Value(#givename.value)
#myscreen_wrapper.setvalue Infield('txtAddress1') Value(#address1.value)
#myscreen_wrapper.setvalue Infield('txtAddress2') Value(#address2.value)
#myscreen_wrapper.setvalue Infield('txtAddress3') Value(#address3.value)
#myscreen_wrapper.setvalue Infield('txtPhone') Value(#phonehme.value)
#myscreen_wrapper.setvalue Infield('txtPostcode') Value(#POSTCODE.value)
* Send the Enter key with the payload
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyEnter)

```

```
Payload(UPDATE_EMPLOYEE)
```

```
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 5250 screen. Otherwise, make the error message appear in a message box on top of the command
```

```
If (#system.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
```

```
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)
#com_owner.Save
Endroutine
* -----
* Handle Termination
* -----
Mthroutine Name(uTerminate) Options(*REDEFINE)
* Clean up the collection of fields on the panel
Invoke Method(#PanelFields.RemoveAll)
* Do any termination defined in the ancestor
Invoke Method(#Com_Ancestor.uTerminate)
Endroutine
End_Com

```

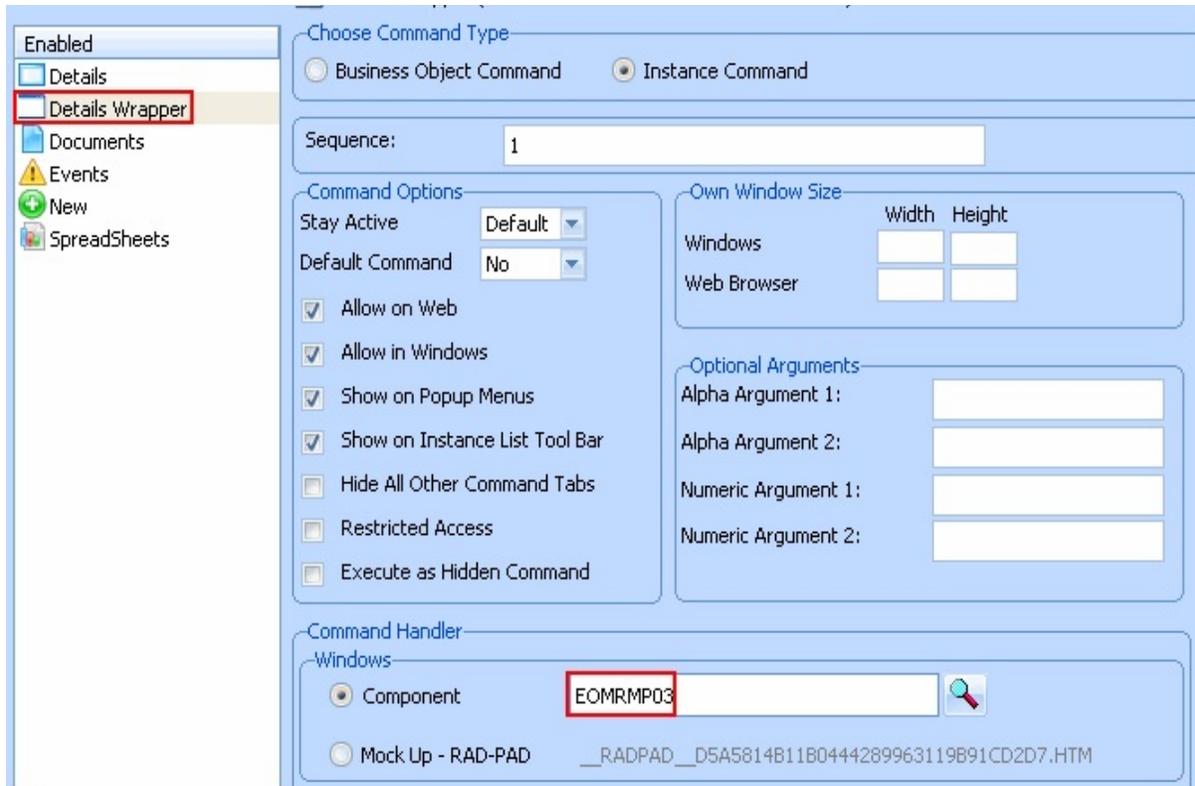
3. Display the Design tab of the component to see the screen wrapper user interface:

The screenshot shows a window titled "EOMRMP03 - DisplayEmployee Wrapper" with a close button in the top right corner. The window contains the following fields and values:

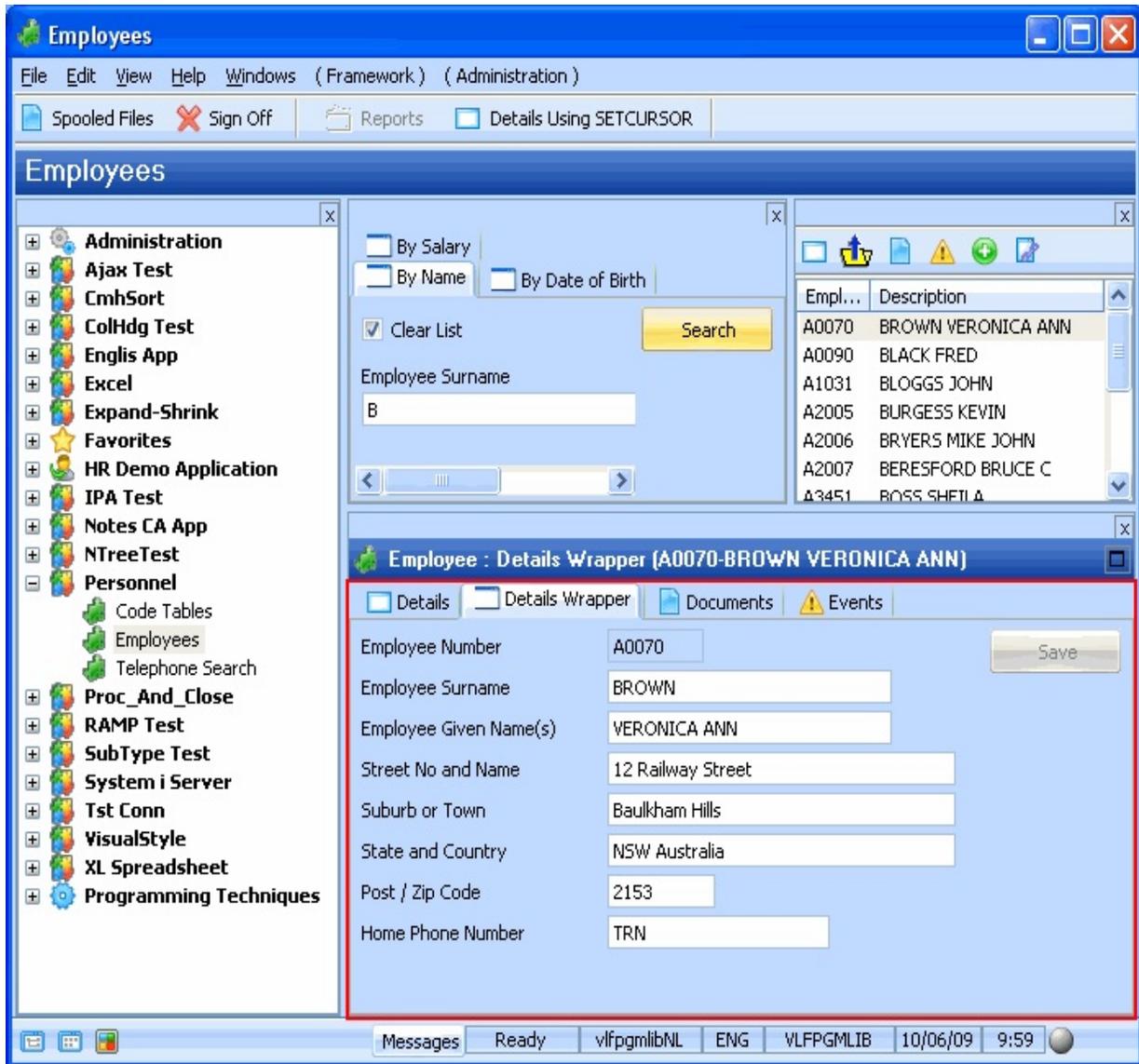
Field Label	Value
Employee Number	ABCDE
Employee Surname	ABCDEFGHJKLMNOPQRST
Employee Given Name(s)	ABCDEFGHJKLMNOPQRST
Street No and Name	aAbBcCdDeEfFgGhHiIjJkKlLm
Suburb or Town	aAbBcCdDeEfFgGhHiIjJkKlLm
State and Country	aAbBcCdDeEfFgGhHiIjJkKlLm
Post / Zip Code	123456
Home Phone Number	ABCDEFGHJKLMNO

A "Save" button is located in the top right corner of the form area.

4. Compile the screen wrapper.
5. In the Framework, display the properties of the Employees business object.
6. Create a new command Details Wrapper, enable it for the Employees business object and associate the iiiRMP03 screen wrapper with it.



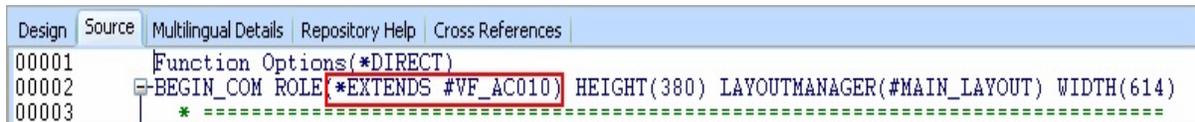
7. Save and restart the Framework.
8. Test your screen wrapper by selecting an employee and displaying the Details Wrapper command handler for an employee:



RAMP-TSAD08 Step 3. Understanding the Screen Wrapper Code

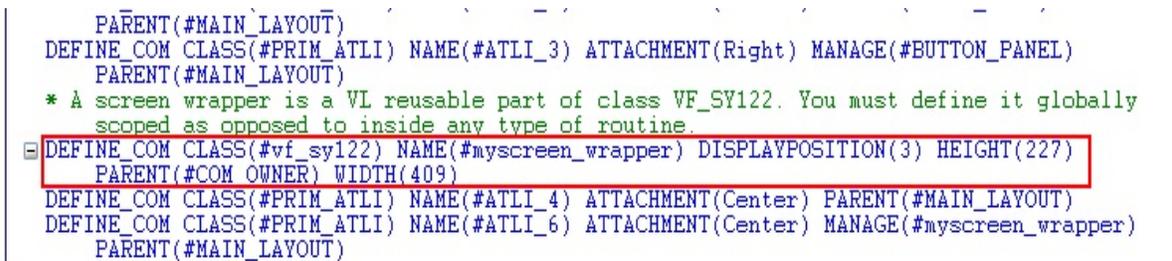
In this step you will examine the code in the screen wrapper.

1. Display the screen wrapper source code in the Visual LANSA editor.
2. The screen wrapper is a command handler, therefore it's ancestor has to be #VF_AC010:



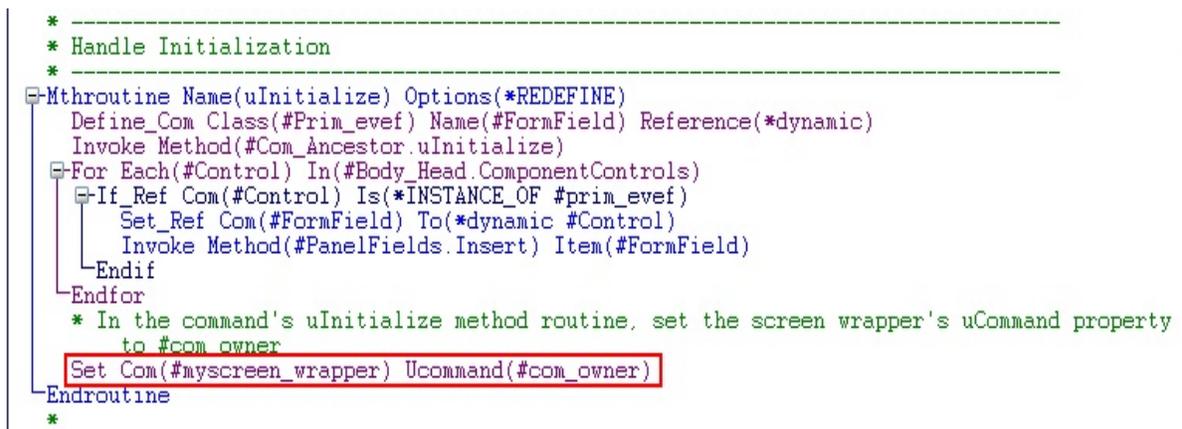
```
Design Source Multilingual Details Repository Help Cross References
00001 Function Options(*DIRECT)
00002 BEGIN_COM ROLE *EXTENDS #VF_AC010 HEIGHT(380) LAYOUTMANAGER(#MAIN_LAYOUT) WIDTH(614)
00003 * =====
```

3. The first thing to note is in the DEFINE_COM statement which defines the screen wrapper component which enables the command handler to communicate with the underlying 5250 screen:



```
PARENT(#MAIN_LAYOUT)
DEFINE_COM CLASS(#PRIM_ATLI) NAME(#ATLI_3) ATTACHMENT(Right) MANAGE(#BUTTON_PANEL)
PARENT(#MAIN_LAYOUT)
* 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) DISPLAYPOSITION(3) HEIGHT(227)
  PARENT(#COM_OWNER) WIDTH(409)
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)
```

4. Next, in the uInitialize even routine, note the statement that sets the screen wrapper's uCommand property to #COM_OWNER :



```
* -----
* Handle Initialization
* -----
Mthroutine Name(uInitialize) Options(*REDEFINE)
  Define_Com Class(#Prim_evef) Name(#FormField) Reference(*dynamic)
  Invoke Method(#Com_Ancestour.Initialize)
  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
  * In the command's uInitialize method routine, set the screen wrapper's uCommand property
  to #com_owner
  Set Com(#myscreen_wrapper) Ucommand(#com_owner)
Endroutine
*
```

You must always set uCommand to #com_owner, otherwise an error message of

type VF_INIT_ERROR will be issued because the screen wrapper will not have access to the command handler.

5. Then notice that RAMP execution is kicked off by the MakerampTSAvailable method in the uExecute method routine:

```
-----  
Mthroutine Name(uExecute) Options(*REDEFINE)  
  Invoke Method(#Com Ancestor.uExecute)  
  #myscreen_wrapper.makerampTSAvailable  
  Set Com(#Save_Button) Enabled(False)  
  #com_owner.enabled := false  
Endroutine
```

6. When RAMP is available, you specify the destination screen to which you want the wrapper to navigate in the #myscreen_wrapper.RampTSAvailable event routine:

```
Evtroutine Handling(#myscreen_wrapper.RampTSAvailable)  
  Invoke Method(#myscreen_wrapper.navigatetoscreen) Name('DisplayEmployee')  
Endroutine
```

7. The vHandleArrive event routine first checks that the screen that has arrived is DisplayEmployee. For error handling, it checks if there is an UPDATE_EMPLOYEE payload (issued when the user clicks the Save button). If there isn't, the wrapper retrieves the values from the fields on the screen and displays them:

```

EvtRoutine Handling(#myscreen_wrapper.vHandleArrive) Arrivedscreen(#CurrentScreen)
  PreviousScreen(#PreviousScreen) Arrivedpayload(#Payload)
  Case (#CurrentScreen)
  -When Value_Is(= 'DisplayEmployee')
    Set Com(#SAVE_BUTTON) Enabled(false)
    * Error handling: Payloads are destroyed when the ARRIVE script finishes executing. Therefore, a payload
      of UPDATE_EMPLOYEE would most likely mean there was a validation error.
  -If (#Payload = UPDATE_EMPLOYEE)
    Use Builtin(message_box_show) With_Args(ok ok info "Validation Error" "Please correct any errors")
  -Else
    * Unlock the framework
    Set Com(#avFrameworkManager) Ulocked(false)
    #myscreen_wrapper.getvalue From('txtEmpno') Value(#empno.value)
    #myscreen_wrapper.getvalue From('txtSurname') Value(#surname.value)
    #myscreen_wrapper.getvalue From('txtGivename') Value(#givename.value)
    #myscreen_wrapper.getvalue From('txtAddress1') Value(#address1.value)
    #myscreen_wrapper.getvalue From('txtAddress2') Value(#address2.value)
    #myscreen_wrapper.getvalue From('txtAddress3') Value(#address3.value)
    #myscreen_wrapper.getvalue From('txtPhone') Value(#phonehme.value)
    #myscreen_wrapper.getvalue From('txtPostcode') Value(#POSTCODE.value)
    #com_owner.enabled := true
  -Endif
  -Endcase
-Endroutine

```

8. Locate the event routine handling the Save button on the screen wrapper which assigns the values on the fields on the wrapper to the fields on the screen and then emulates the pressing of the Enter key:

```

* -----
* Handle Save
* -----
Mthroutine Name(Save)
  * Set the 5250 field values to the values from this panel
  #myscreen_wrapper.setvalue Infield('txtSurname') Value(#surname.value)
  #myscreen_wrapper.setvalue Infield('txtGivename') Value(#givename.value)
  #myscreen_wrapper.setvalue Infield('txtAddress1') Value(#address1.value)
  #myscreen_wrapper.setvalue Infield('txtAddress2') Value(#address2.value)
  #myscreen_wrapper.setvalue Infield('txtAddress3') Value(#address3.value)
  #myscreen_wrapper.setvalue Infield('txtPhone') Value(#phonehme.value)
  #myscreen_wrapper.setvalue Infield('txtPostcode') Value(#POSTCODE.value)
  * Send the Enter key with the payload
  #myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyEnter) Payload(UPDATE_EMPLOYEE)
-Endroutine

```

9. Lastly have a look at the routine handling the #myscreen_wrapper.RampMessage event to see how RAMP-TS can handle different types of errors in a screen wrapper:

```

* Listen to messages from RAMP and the 5250 application
Evroutine 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 5250 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
-Otherwise
  Use Builtin(message_box_show)
  With_Args(ok ok info *Component ('Unknown message type ' + #MsgType + 'encountered'))
  The result of executing expression ('Unknown message type ' + #MsgType + 'encountered') will be checked to
-Endcase
Endroutine

```

RAMP-TSAD08 Step 4. Test Updating the Screen from the Wrapper

In this step you will

1. Display the details of an employee in the Details Wrapper in the Framework.
2. Turn the application trace on from the Framework menu
3. Make a change to, for example, the employee surname and press the Save button.
4. Examine the trace. Notice how the screen wrapper first assigns the field values on the wrapper to the fields on the underlying 5250 screen and sends the Enter key. RAMP then navigates to the FindEmployee screen which updates the instance list and then navigates back to the DisplayEmployee screen.

The screenshot shows the 'Application Level Trace Details' window with a table of events. The table has three columns: Component, Window, and Event. The events describe the process of updating an employee record, including setting field values, sending the Enter key, and navigating between screens.

Component	Window	Event
EOMRMP03	MAIN	SETVALUE of txtSurname index = 0 to value BLOGGS (type = string) .Current form is DisplayEmployee (15:38:20:18)
EOMRMP03	MAIN	SETVALUE of txtSurname completed. Current form is DisplayEmployee (15:38:20:18)
EOMRMP03	MAIN	SETVALUE of txtGivenname index = 0 to value JOHN ALAN (type = string) .Current form is DisplayEmployee (15:38:20:33)
EOMRMP03	MAIN	SETVALUE of txtGivenname completed. Current form is DisplayEmployee (15:38:20:33)
EOMRMP03	MAIN	SETVALUE of txtAddress1 index = 0 to value 3 Woodbury Road (type = string) .Current form is DisplayEmployee (15:38:20:49)
EOMRMP03	MAIN	SETVALUE of txtAddress1 completed. Current form is DisplayEmployee (15:38:20:49)
EOMRMP03	MAIN	SETVALUE of txtAddress2 index = 0 to value Winston Hills (type = string) .Current form is DisplayEmployee (15:38:20:49)
EOMRMP03	MAIN	SETVALUE of txtAddress2 completed. Current form is DisplayEmployee (15:38:20:49)
EOMRMP03	MAIN	SETVALUE of txtAddress3 index = 0 to value NSW Australia (type = string) .Current form is DisplayEmployee (15:38:20:64)
EOMRMP03	MAIN	SETVALUE of txtAddress3 completed. Current form is DisplayEmployee (15:38:20:64)
EOMRMP03	MAIN	SETVALUE of txtPhone index = 0 to value (02) 9668 9235 (type = string) .Current form is DisplayEmployee (15:38:20:80)
EOMRMP03	MAIN	SETVALUE of txtPhone completed. Current form is DisplayEmployee (15:38:20:80)
EOMRMP03	MAIN	SETVALUE of txtPostcode index = 0 to value 2100 (type = string) .Current form is DisplayEmployee (15:38:20:80)
EOMRMP03	MAIN	SETVALUE of txtPostcode completed. Current form is DisplayEmployee (15:38:20:80)
EOMRMP03	MAIN	SENDKEY Enter requested. Current form is DisplayEmployee (15:38:20:96)
EOMRMP03	MAIN	Note => A payload accompanied this SENDKEY operation. The value was UPDATE_EMPLOYEE (15:38:20:96)
EOMRMP03	MAIN	Posting screen. Current form is DisplayEmployee (15:38:20:96)
EOMRMP03	MAIN	Executing vHandle_DEPART function in form DisplayEmployee (15:38:20:96)
EOMRMP03	MAIN	Execution of script associated with DisplayEmployee completed. No error detected. (15:38:20:96)
EOMRMP03	MAIN	SENDKEY Enter completed. Server response will be asynchronous. Your script should end now. Current form is DisplayEmp
EOMRMP03	MAIN	Form with name "FindEmployee" has arrived. (15:38:20:439)
EOMRMP03	MAIN	Executing vHandle_ARRIVE function in form FindEmployee (15:38:20:439)
EOMRMP03	MAIN	Note => A payload accompanied this request. The value was UPDATE_EMPLOYEE (15:38:20:439)
EOMRMP03	MAIN	AVSIGNAL executed with WithId= Update_List_Entry To= BUSINESSOBJECT SendAInfo1= A1031 SendAInfo2= Se
EOMRMP03	MAIN	Signalling avEvent WithId= Update_List_Entry Wait= FALSE To= BUSINESSOBJECT SendAInfo1= A1031
EOMRMP03	MAIN	NAVIGATE_TO_SCREEN invoked to navigate to screen DisplayEmployee (15:38:20:439)
EOMRMP03	MAIN	Preparing a navigation plan to get to screen DisplayEmployee (15:38:20:439)
EOMRMP03	MAIN	An optimized pre-existing navigation plan from FindEmployee to DisplayEmployee will be used. (15:38:20:439)

At the bottom of the window, there are two buttons: 'Save Trace to File' and 'Clear Trace'.

Summary

Important Observations

Before you create a screen wrapper, name all the fields on the 5250 screen(s) that will be used in the wrapper

You also need to ensure there is a navigation path to and from the destination screen to be wrapped.

A screen wrapper is a Visual LANSA reusable part with Ancestor #VF_AC010 (command handler) which defines class #VF_SY122.

In the screen wrapper, you start RAMP execution by invoking the MakerampTSAvailable event. Once RAMP is running, you specify the destination screen the wrapper is to navigate to in the routine handling RampTSAvailable event.

You use the ScreenWrapper.GetValue and ScreenWrapper.SetValue methods to pass values between the 5250 screens and the wrapper.

You snap the screen wrapper into the Framework as you would snap in any non-RAMP command handler (using the business object properties, not RAMP Tools)

What You Should Know

How to create a basic screen wrapper

RAMP-TSAD09: Screen Wrapper with a Subfile

In this tutorial you will enhance your screen wrapper to include a photo of the employee and the skills subfile presented as a list view.

Objectives

Learn how to handle subfiles as a list view in screen wrapper

Learn how easy it to wrap a screen in a more intuitive interface

In this tutorial you will enhance the screen wrapper to show a photo and the skills subfile as a list view.

If you compare this simple wrapper with the RAMPed 5250 screen (the Details tab), you can see how easy it is to make a basic screen more informative.

Acquired	Skill Code	Skill	Comment	Gra...
30/06/96	ADMIN1	Administra...	Met require...	P
1/03/98	ADMIN2	Administra...		P
10/12/95	ADVPGM	Advanced ...		P
4/05/98	COM	Communic...		D
5/06/90	CS	Computer ...	Very good re...	D
20/02/90	INDUCT	Company I...		P
5/02/98	INTRO	Company I...	Met require...	P
5/02/98	KEY	Keyboard ...	Met require...	P
10/12/97	MANAGE1	Manageme...	Good result	M
1/03/98	MANAGE2	Manageme...	Met require...	P
10/12/97	MARKET1	Marketing ...	Excellent	D
3/05/98	MARKET2	Marketing ...		P

To achieve this objective, you will complete the following steps:

- [RAMP-TSAD09 Step 1. Add an Image to the Screen Wrapper](#)
- [RAMP-TSAD09 Step 2. Add Skills List View to the Wrapper](#)
- [RAMP-TSAD09 Step 3. Add Code to Populate the List View](#)

- [Summary](#)

Before You Begin

In order to complete this tutorial, you must have completed [RAMP-TSAD08: Screen Wrapper Basics](#).

Note About This Tutorial

On some systems the screen wrapper created in this tutorial causes a 'multiple sendkey' error. The problem is currently being investigated. A work-around is to name the screen with a different name when it is in update mode, for example 'UpdateEmployee' instead of using the F21 keypress in the arrival script to distinguish the screen in display and update mode.

RAMP-TSAD09 Step 1. Add an Image to the Screen Wrapper

In this step you will add an image to the Browse/Maintain Employee and Skill Files screen wrapper.

In this example you will be using a standard .gif file shipped with the Framework, in a real application you would have the employee photos stored in a file on the server and you would retrieve them with the other employee details.

1. Display the iiiRMP03 screen wrapper in the Visual LANSA editor.
2. Display the Design tab.
3. Drag an image control from the Common Controls tab in Favorites to the screen.

Notice that it is automatically placed under the fields in the screen wrapper by the layout manager that manages the screen. In this exercise you want to position the image manually.

4. Display the Source tab and locate the DEFINE_COM statement for the image, and the layout item created for it.
5. Comment out the statement defining the flow layout item.

```
DEFINE_COM CLASS(#PRIM_IMG) NAME(#IMGE_1) DISPLAYPOSITION(9) HEIGHT(117) LEFT(4) PARENT(#BODY_HEAD) TABPOSITION(9)
TOP(188) WIDTH(108)
*DEFINE_COM CLASS(#PRIM_FWLI) NAME(#FWLI_1) MANAGE(#IMGE_1) PARENT(#BODY_HEAD_FLOW)
```

6. Switch to the Design tab and move and position the image like this:



7. Display the Source tab and add this statement to assign an image file to the image control after the values for fields on the screen wrapper have been retrieved:

```
Set Com(#imge_1) Filename('C:\Program Files\LANSA\LANSA\Imports\VLFRAMEWORK\df_im001.gif')
```

You need to replace the path information with the path in your system where df_im001.gif is located. Your code should look like this:

```
Else
  #com_owner.enabled := true
  #myscreen_wrapper.getvalue From('txtEmpno') Value(#empno.value)
  #myscreen_wrapper.getvalue From('txtSurname') Value(#surname.value)
  #myscreen_wrapper.getvalue From('txtGivename') Value(#givename.value)
  #myscreen_wrapper.getvalue From('txtAddress1') Value(#address1.value)
  #myscreen_wrapper.getvalue From('txtAddress2') Value(#address2.value)
  #myscreen_wrapper.getvalue From('txtAddress3') Value(#address3.value)
  #myscreen_wrapper.getvalue From('txtPhone') Value(#phonehme.value)
  #myscreen_wrapper.getvalue From('txtPostcode') Value(#POSTCODE.value)
  Set Com(#imge_1)
  Filename('C:\Program Files\LANSP5Final\LANSA\Imports\VLFRAMEWORK\df_im001.gif')
Endif
```

8. Compile the screen wrapper. If the compile fails because the wrapper is being used, restart the Framework.
9. In the Framework, display the Details Wrapper for an employee:

Employee : Details Wrapper (A0070-BROWN VERONICA ANN)

Details Details Wrapper Documents Events

Employee Number	A0070	 <input type="button" value="Save"/>
Employee Surname	BROWN	
Employee Given Name(s)	VERONICA ANN	
Street No and Name	12 Railway Street	
Suburb or Town	Baulkham Hills	
State and Country	NSW Australia	
Post / Zip Code	2153	
Home Phone Number	TRN	

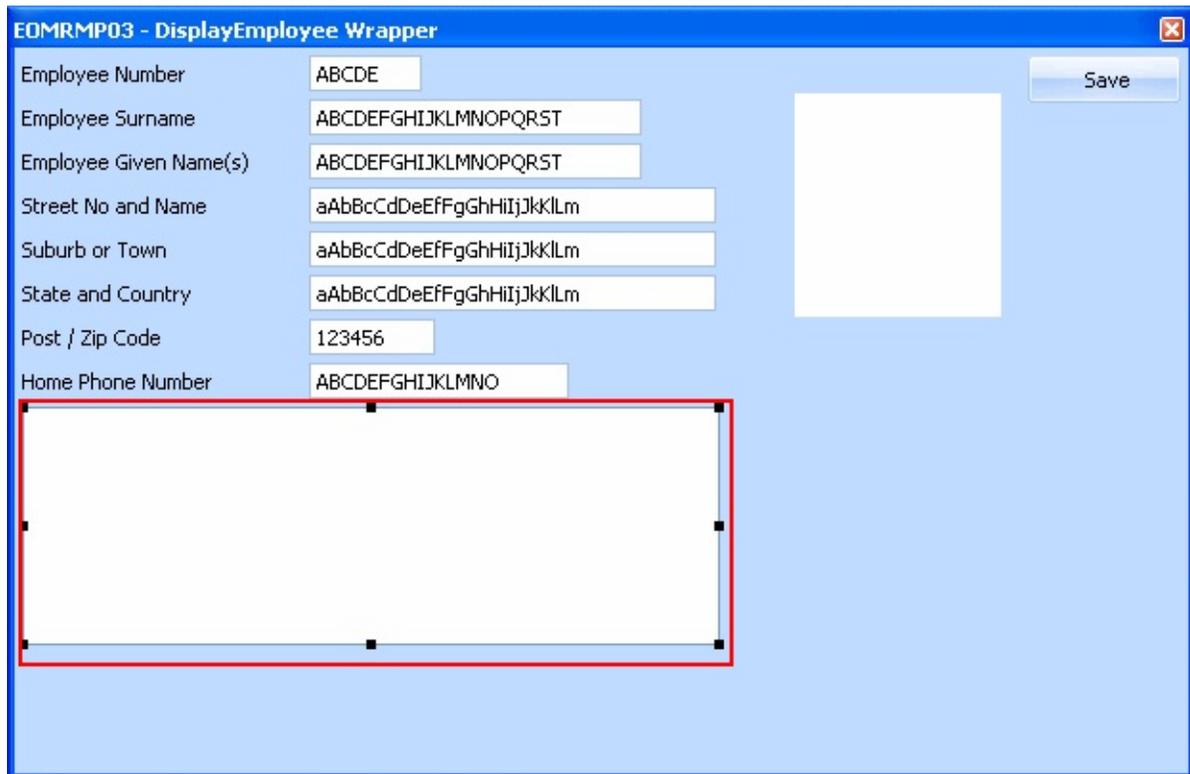
10. Shut down the Framework.

RAMP-TSAD09 Step 2. Add Skills List View to the Wrapper

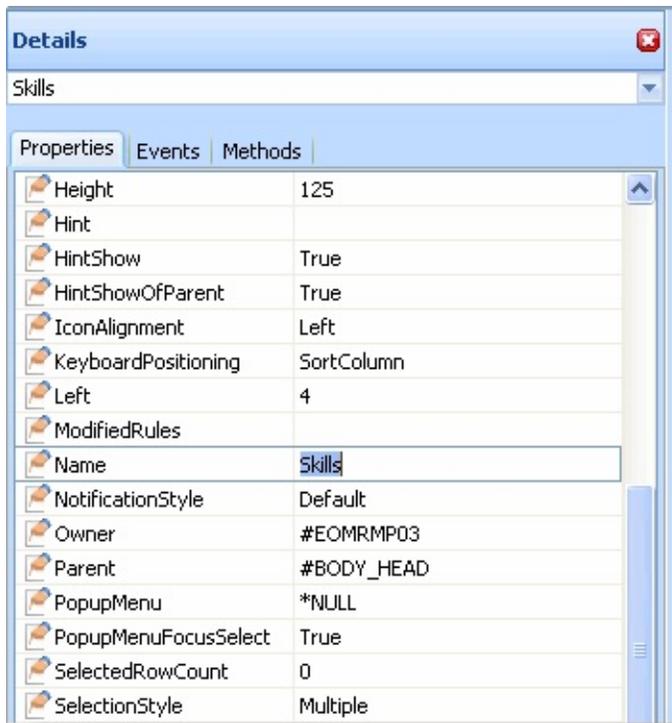
In this step you will add a list view to the screen wrapper that shows the skills subfile on the Browse/Maintain Employee and Skill Files screen.

Note that you named the subfile columns and the plus sign in [RAMP-TSAD08 Step 1. Name the Fields to Be Used in the Wrapper](#).

1. Display the screen wrapper user interface in the Visual LANSA editor.
2. Drag a list view control to the screen wrapper and size it like this:



3. Name it Skills.



4. Locate the PLSKL file in the repository, and drag the DATEACQ and SKILCODE fields in the file to the list view.
5. Locate the SKLTAB file in the repository and drag the SKILDESC field to the list view.
6. Lastly add the COMMENT and GRADE fields from the PLSKL file to the list view.

Your list view should look like this:

EOMRMP03 - DisplayEmployee Wrapper

Employee Number: ABCDE

Employee Surname: ABCDEFGHIJKLMNOPQRST

Employee Given Name(s): ABCDEFGHIJKLMNOPQRST

Street No and Name: aAbBcCdDeEfFgGhHiIjJkKlLm

Suburb or Town: aAbBcCdDeEfFgGhHiIjJkKlLm

State and Country: aAbBcCdDeEfFgGhHiIjJkKlLm

Post / Zip Code: 123456

Home Phone Number: ABCDEFGHIJKLMNO

Save

Acquired	Skill Code	Skill	Comment	Grade Obt
✓ AB...	ABCDE...	ABCDEFG...	aAbBcCdDeEfFg...	A

7. Save the wrapper.

RAMP-TSAD09 Step 3. Add Code to Populate the List View

In this step you will add the code to populate the list view.

1. Display the source code of the screen wrapper.
2. Locate the #myscreen_wrapper.RampTSAvailable event routine and add this code to clear the fields and the subfile on it when the screen arrives:

```
Clr_List Named(#skills)  
#XG_HEAD := *null
```

Your code should look like this:

```
Evtroutine Handling(#myscreen_wrapper.RampTSAvailable)  
  Clr_List Named(#skills)  
  #XG_HEAD := *null  
  
  Invoke Method(#myscreen_wrapper.navigatetoscreen) Name('DisplayEmployee')  
Endroutine
```

3. Add the following method routine to the end of your screen wrapper source:

```
Mthroutine Name(uGetSubfilePage)  
  
Define_Map For(*result) Class(#prim_boln) Name(#NextPage)  
Define Field(#MoreVal) Type(*char) Length(1)  
  
#listcount := 1  
#NextPage := false  
  
Dowhile (#myscreen_wrapper.check_field_exists( "dateacq" #listcount ))  
  
#myscreen_wrapper.getvalue From("dateacq") Value(#vf_eltxts)  
Defaultvalue(#ddmmyy) Index(#listcount)  
#myscreen_wrapper.getvalue From("skillcode") Value(#skilcode)  
Index(#listcount)  
#myscreen_wrapper.getvalue From("skilldesc") Value(#skilldesc)  
Index(#listcount) Defaultvalue("Default value")  
#myscreen_wrapper.getvalue From("comment") Value(#comment)  
Index(#listcount)  
#myscreen_wrapper.getvalue From("grade") Value(#grade) Index(#listcount)
```

```

* You can put some tracing
#com_owner.avframeworkmanager.avRecordTrace Component(#com_owner)
Event("Adding entry = " + #vf_eltxts + ", " + #skilcode + ", " + #skildesc + ",
" + #grade)

Add_Entry To_List(#skills)

#listcount += 1

Endwhile

* when identifying this screen we set the name of the "+" sign =
"moreindicator". The presence of that field in the last row of the subfile tells
us whether there is another page. The last row is one less than the current
value of #listcount.

#listcount -= 1

If (#myscreen_wrapper.check_field_exists( "moreindicator" #listcount ))
#myscreen_wrapper.getvalue From("moreindicator") Value(#MoreVal)
Index(#listcount)
#NextPage := (#MoreVal.trim *NE "")
Endif

Endroutine

```

This code traverses the skills subfile by first checking if the first column corresponding to the DATEACQ field exists. If it is present on the screen, the routine gets the values for all the subfile columns.

Fields in RAMP-TS subfiles are indexed starting from 1. A subfile page with 7 rows will have 7 instances of each of the fields in the subfile. Here we increase #listcount and use it to get the value of a field.

Note that you can specify a default value when using the getvalue method. The moreindicator is the name you gave to the plus sign on the screen.

4. Now locate the #myscreen_wrapper.vHandleArrive event routine.

5. Define a Boolean class #MoreRecords which will be used by the wrapper to determine if there are more records in the skills subfile.

```
Define_Com Class(#prim_boln) Name(#MoreRecords)
```

6. Then add this code to check value returned by the uGetSubfilePage method, and if there are more records, to send a PageDown keystroke to get to the next subfile page.

```
#MoreRecords := #com_owner.uGetSubfilePage  
If (#MoreRecords)  
if (#previouscreen = 'DisplayEmployee')  
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyPageDown)
```

(The check for the previous screen is there simply to slow the screen wrapper down so that repeated SENDKEY requests on fast machines get queued properly.)

Your code should look like this:

```

EvtRoutine Handling(#myscreen_wrapper.vHandleArrive) Arrivedscreen(#CurrentScreen)
    PreviousScreen(#PreviousScreen) Arrivedpayload(#Payload)
    Define_Com Class(#prim_boln) Name(#MoreRecords)
    Case (#CurrentScreen)
        When Value_Is(= 'DisplayEmployee')
            Set Com(#SAVE_BUTTON) Enabled(false)
            If (#Payload = UPDATE_EMPLOYEE)
                Use Builtin(message_box_show)
                With_Args(ok ok info "Validation Error" "Please correct any errors")
            Else
                #MoreRecords := #com_owner.uGetSubfilePage
                If (#MoreRecords)
                    if (#previousScreen = 'DisplayEmployee')
                        #myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyPageDown) Payload(UPDATEMODE)
                    else
                        Set Com(#avFrameworkManager) Ulocked(false)
                        #com_owner.enabled := true
                        #myscreen_wrapper.getvalue From('txtEmpno') Value(#empno.value)
                        #myscreen_wrapper.getvalue From('txtSurname') Value(#surname.value)
                        #myscreen_wrapper.getvalue From('txtGivenname') Value(#givenname.value)
                        #myscreen_wrapper.getvalue From('txtAddress1') Value(#address1.value)
                        #myscreen_wrapper.getvalue From('txtAddress2') Value(#address2.value)
                        #myscreen_wrapper.getvalue From('txtAddress3') Value(#address3.value)
                        #myscreen_wrapper.getvalue From('txtPhone') Value(#phonehme.value)
                        #myscreen_wrapper.getvalue From('txtPostcode') Value(#POSTCODE.value)

                        Set Com(#inge_1)
                        Filename('C:\Program Files\LANSP5Final\LANSA\Imports\VLFRAMEWORK\df_im001.gif')
                    Endif
                endif
                #com_owner.enabled := true
            Endif
        Endif
    Endcase
Endroutine

```

7. Compile the screen wrapper.
8. Start the Framework and display the Details Wrapper command handler for an employee.

Employee : Details Wrapper [A0070-BROWN VERONICA ANN]

Details
 Details Wrapper
 Documents
 Events

Employee Number: A0070
 Save

Employee Surname: BROWN

Employee Given Name(s): VERONICA ANN

Street No and Name: 12 Railway Street

Suburb or Town: Baulkham Hills

State and Country: NSW Australia

Post / Zip Code: 2153

Home Phone Number: TRN



Acquired	Skill Code	Skill	Comment	Grade Obtai...
25/03/98	ADMIN1	Administratn ...	Met req...	D
1/05/98	ADMIN2	Administratn ...	G	P
4/05/98	COM	Communicati...		D
5/05/98	CS	Computer Sci...		P
30/06/96	ADMIN1	Administratn ...	Met req...	P
1/03/98	ADMIN2	Administratn ...		P
10/12/95	ADVPGM	Advanced Pr...		P
4/05/98	COM	Communicati...		D

Summary

Important Observations

You can make the screen wrapper to show subfile records in a list view to eliminate the need to page down.

-

What You Should Know

How to enhance your screen wrapper by adding components such as images

How to display a subfile as a list view on a screen wrapper.

Scripting

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

[Learning](#)

[Using](#)

[Debugging](#)

Learning

[Anatomy of Scripts](#) introduces you to RAMP scripts.

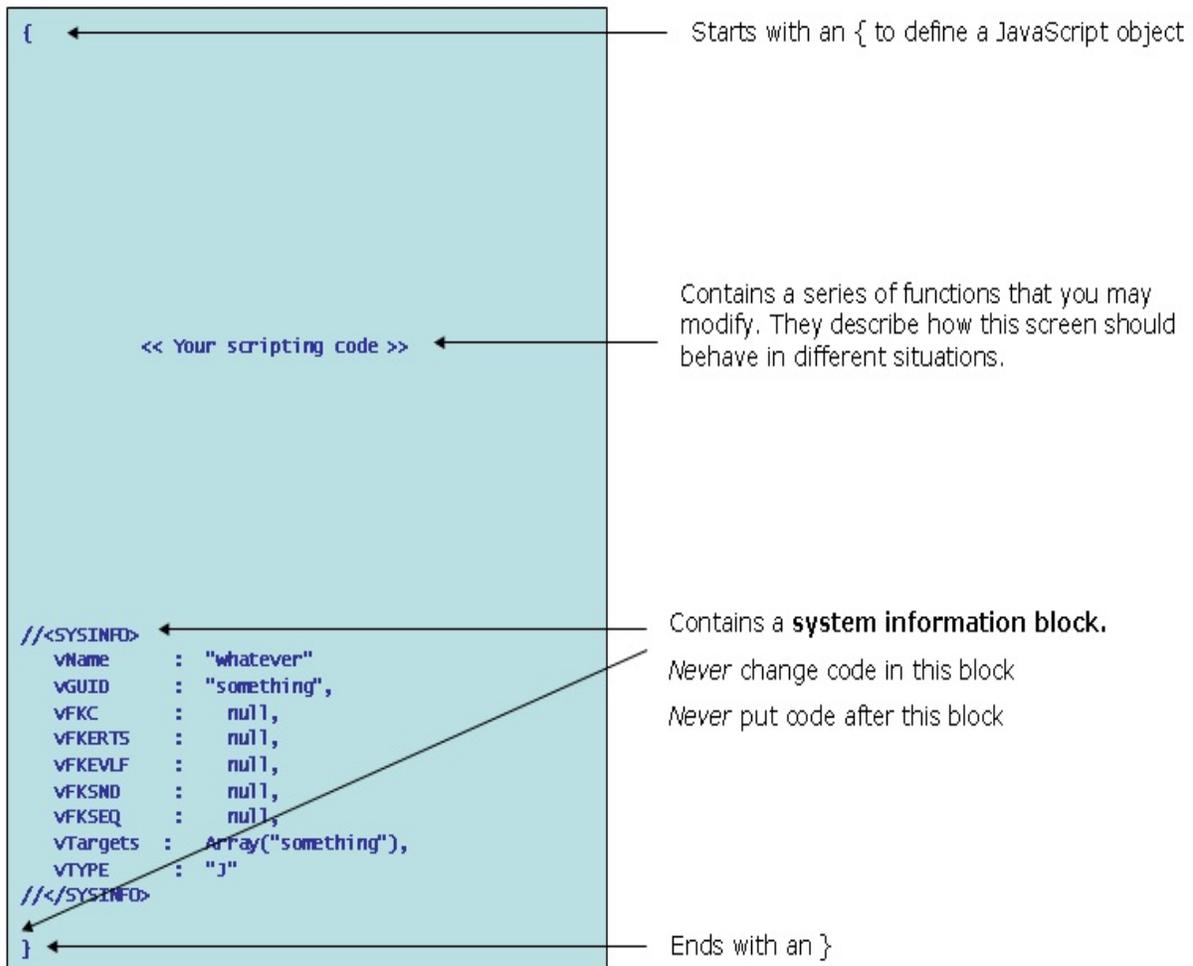
[Javascript Essentials](#) teaches you some basic techniques you will often use when writing scripts.

Application level tracing

Use the Tracing option in the Framework menu to start [Application Level Tracing](#)

Anatomy of Scripts

Every classified screen has a single script associated with it. The script is always structured like this:



[Special Screen Script](#)

[Junction Screen Script](#)

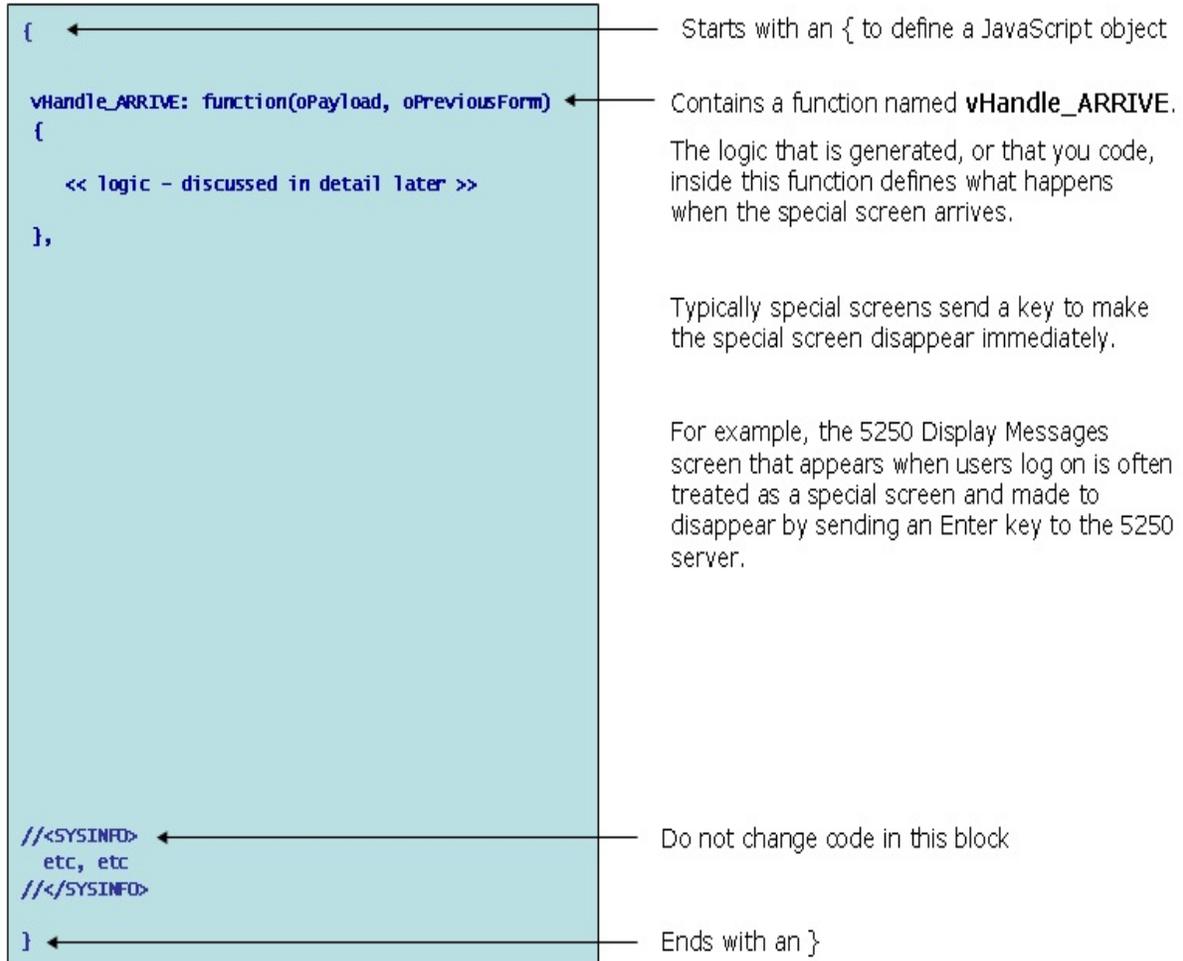
[Destination Screen Script](#)

[vHandle_ARRIVE Function](#)

[Your RAMP-TS Screen Script Defines a JavaScript Object](#)

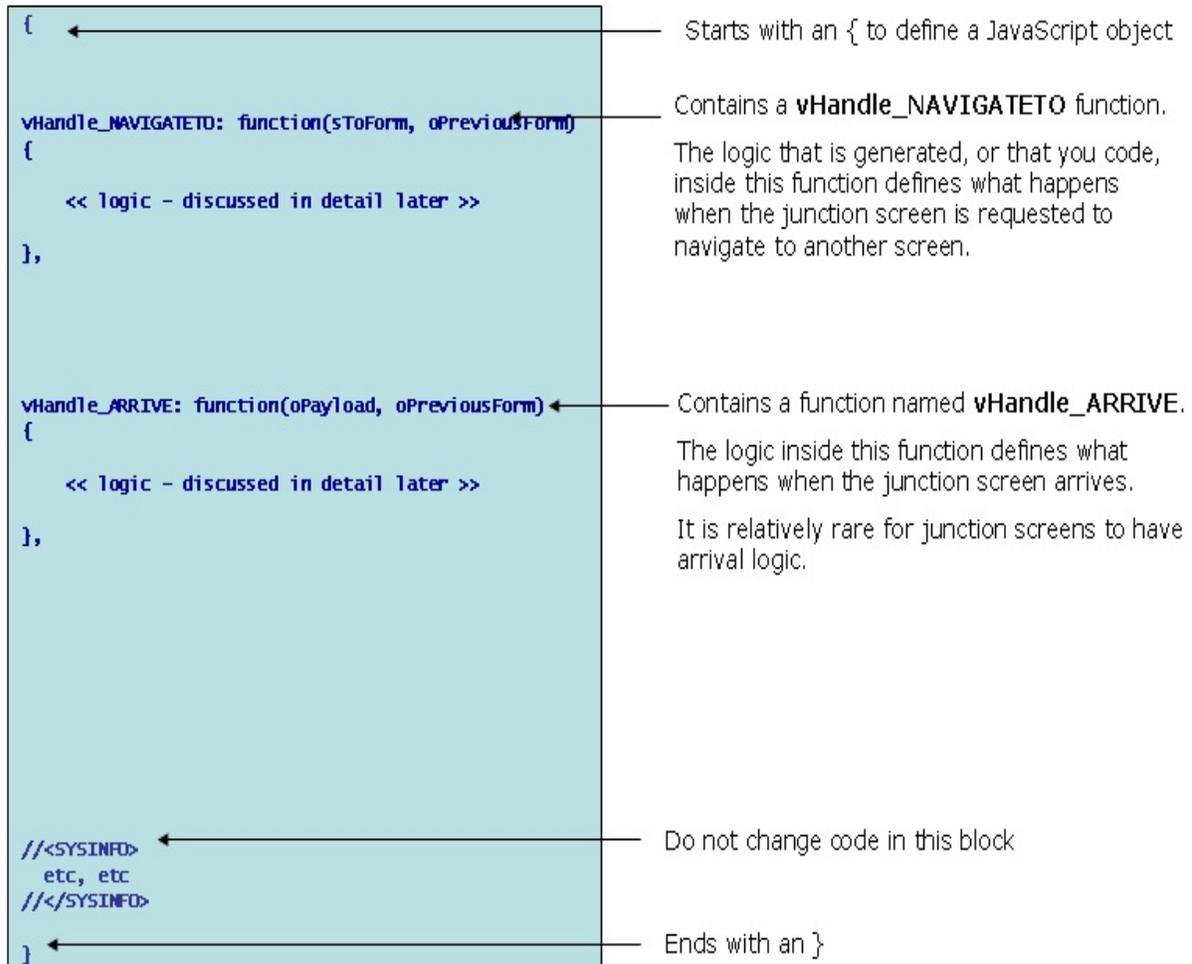
Special Screen Script

The script associated with a **SPECIAL** screen is typically structured like this:



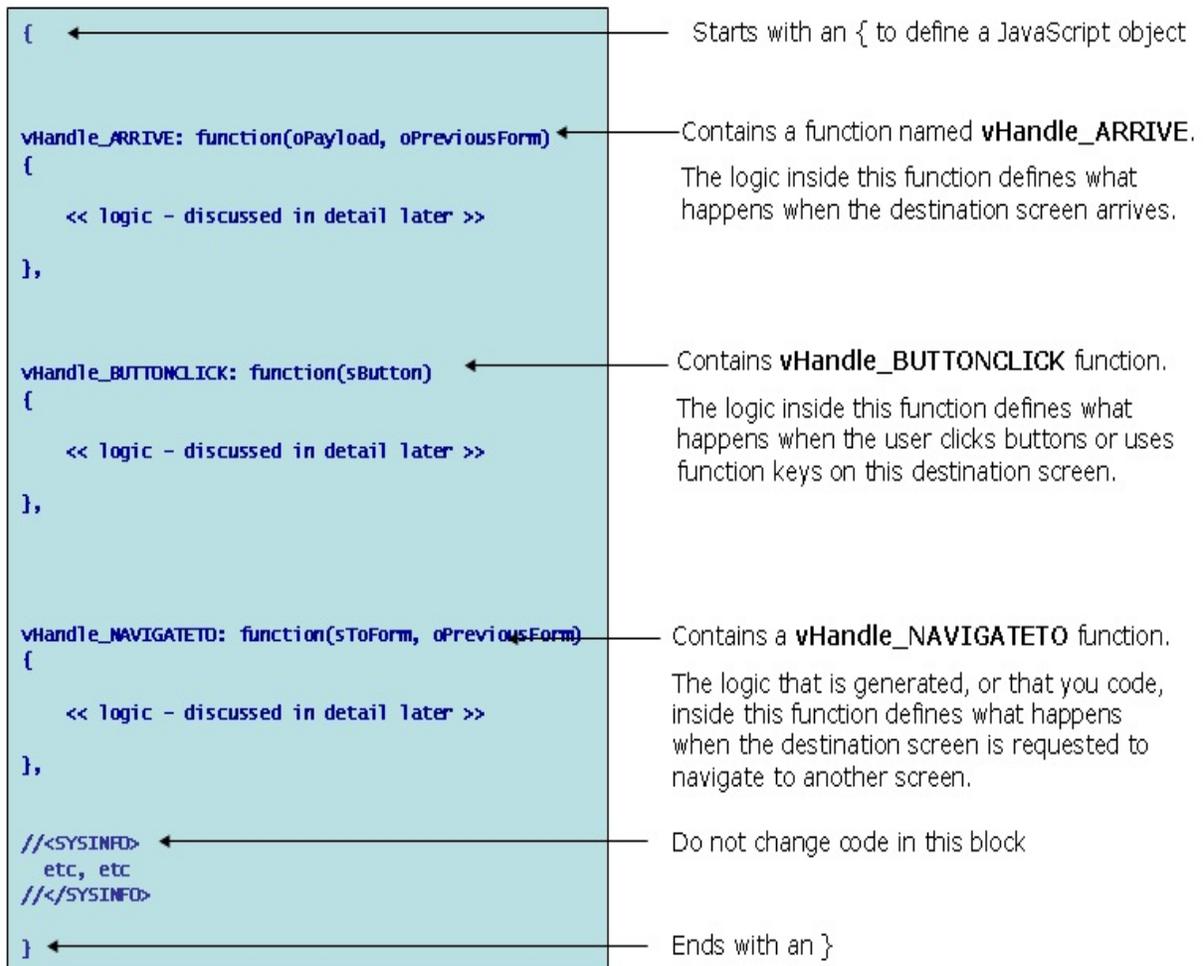
Junction Screen Script

The script associated with a **JUNCTION** screen is typically structured like this:



Destination Screen Script

The script associated with a **DESTINATION** screen is typically structured like this:



vHandle_NAVIGATETO Function

This is the **vHandle_NAVIGATETO** function for an example screen named "JUNCTION_A". It services navigation requests made to it by the RAMP framework. Imagine "JUNCTION_A" can handle requests to navigate to "DESTINATION_B" and to "JUNCTION_C":

```
vHandle_NAVIGATETO: function(sToForm, oPreviousForm)
{
  var bReturn = true;

  HIDE_CURRENT_FORM();
  SETBUSY(true);

  switch (sToForm)
  {
    /* <NAVIGATESWITCH> - Do not remove or alter this line */

    case "DESTINATION_B"
    {
      SETVALUE("menuoption","2");
      SENDKEY(KeyEnter);
      Q_CHECK_CURRENT_FORM("DESTINATION_B", ...etc...)
    }
    break;

    case "JUNCTION_C"
    {
      SETVALUE("menuoption","47");
      SENDKEY(KeyEnter);
      Q_CHECK_CURRENT_FORM("JUNCTION_C", ...etc...)
    }
    break;

    <etc, etc>
  }

  return(bReturn);
},
```

Hide the current framework form to make any navigations invisible and set the framework in a busy state

Never remove or reposition this special tag line. It tells RAMP Tools where to inject new navigations into this script.

Handle a request to navigate to the screen named "DESTINATION_B" by setting the menu option to 2, sending key enter and checking that the correct form arrives as a result.

Handle a request to navigate to the screen named "JUNCTION_C" by setting the menu option to 47, sending key enter and checking that the correct form arrives as a result.

Return the a boolean value back to the RAMP framework indicating success or failure

vHandle_ARRIVE Function

This is the **vHandle_ARRIVE** for an example screen named "DESTINATION_B". It executes whenever "DESTINATION_B" is displayed:

<pre>vHandle_ARRIVE: function(oPayload, oPreviousForm) { var bReturn = true; SHOW_CURRENT_FORM(true); HIDE_5250_BUTTONS(); SETBUSY(false); /* <ARRIVE /> - Do not remove or alter this line */ return(bReturn); },</pre>	<p>← vHandle_ARRIVE functions receives an optional payload and a reference to the previously displayed screen. More about these later.</p> <p>← Show the current framework RAMP screen and show the function keys as buttons in the framework.</p> <p>← Hide any buttons visible on the 5250 screen</p> <p>← Drop the busy state, allowing user interaction</p> <p>← Never remove or reposition this special tag line. It tells RAMP Tools where to inject new code into this script.</p> <p>← Return the a boolean value back to the RAMP framework indicating success or failure</p>
---	--

Your RAMP-TS Screen Script Defines a JavaScript Object

The script you create for each RAMP-TS screen defines a JavaScript object.

Like any JavaScript object it has functions and properties.

The standard shipped RAMP-TS functions and properties are as follows (commonly used features are shown in bold):

Name	Type	Modifiable	Description/Comments
vFKC	String Array	No	Function key ca
vFKERTS	String	No	Function key enablement for screen
vFKEVLF	String	No	Function key/bu enablement for RAMP-TS
vFKSEQ	String	No	Function key se numbers
vFKSND	String Array	No	Function keys to to 5250 server
vGUID	String	No	GUID of the scri definition
vHandle_ARRIVE	Function	N/A	Screen arrival handling function
vHandle_AVEVENT	Function	N/A	Screen VLF eve handling function
vHandle_BUTTONCLICK	Function	N/A	Screen button c function key ha function

vHandle_DEPART	Function	N/A	Screen departur handling functio
vHandle_NAVIGATETO	Function	N/A	Screen navigati handling functio
vHandle_USER_NAVIGATION_PLAN	Function	N/A	Screen navigati override functio
vLastMessage	String	No	Latest message have arrived on screen
vLatestVariant	String	No	Latest screen va to have arrived
vName	String	No	Name of the scr
vOverrideExitJunction	String	Yes	Override of exit junction associa with this screen
vTargets	String Array	No	Screens that this screen can naviq
vTYPE	String	No	Type of screen

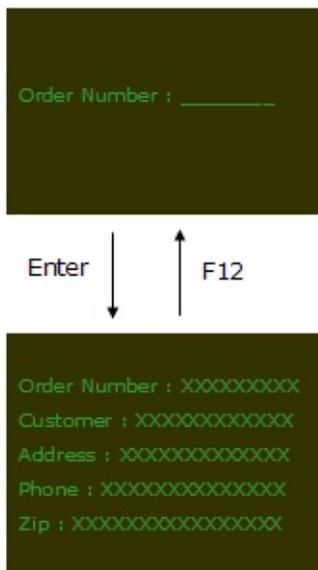
You can add your own functions and properties.

Note: Do not prefix your own functions and properties with "v" (lowercase) or "V" (uppercase). The v*/V* namespaces are reserved by the RAMP-TS product to allow for the future expansion of the standard shipped functions and properties.

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:



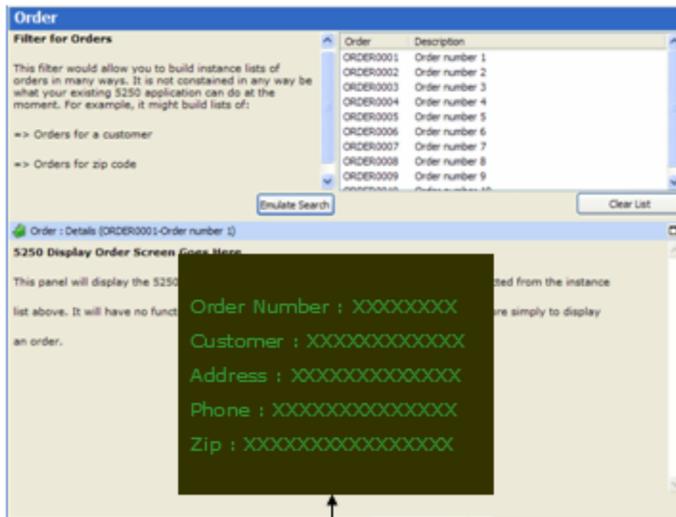
The user repeats order inquiries by using the F12 function key.

Modernized Version

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

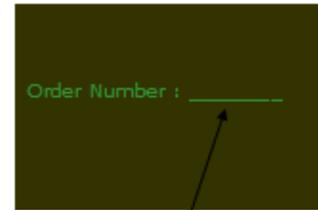
The GETORDER screen becomes a junction screen. It will not be displayed.

The SHOWORDER screen becomes a destination screen.



SHOWORDER
DESTINATION 5250 SCREEN

GETORDER
JUNCTION 5250 SCREEN



ORDERNUMBER field

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

GETORDER has a script with a [vHandle_NAVIGATETO Function](#) which shows the SHOWORDER screen.

The SHOWORDER screen has a script with a:

[vHandle_NAVIGATETO Function](#) which contains code to cancel out of the SHOWORDER screen.

A [vHandle_BUTTONCLICK](#) function 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 [vHandle_NAVIGATETO](#) function is executed to return to the navigation network. Once there, the navigation path to display the SHOWORDERS screen with the details of the selected order is built.

See the tutorial [RAMP-TS009: Tracing Navigation](#) for details of this type of navigation.

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 LANSAs for the Web and HTML manipulation.

This section describes some Javascript essentials:

[External JavaScript Documentation](#)

[Basic Javascript syntax](#)

[Reading, Writing and Storing Values](#)

[alert\(\)](#)

[Converting Numbers to Strings](#)

[Converting String to Numbers](#)

[String Manipulation Functions](#)

[Is This Variable Number or String?](#)

[JavaScript Coding Styles](#)

[Using the objGlobal Object](#)

External JavaScript Documentation

Click here to access 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).

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
}
```

Reading, Writing and Storing Values

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

[Scripting Pop-up Menu](#)

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 (+)

```
var S1 = "Customer";  
var S2 = "123456";  
var S3 = S1 + " " + S2 +  
"could not be found";
```

puts Customer 123456could not be found in variable S3.

IndexOf – finds first occurrence of a string in a string

```
/*    012345678901 */  
var S1 =  
"ABCDHELLOABC";  
var pos =  
S1.indexOf("HELLO");
```

will put the number 4 into variable pos.

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"

length – returns the length of a string

into S3.

```
/* 012345678901 */  
var S1 =  
"ABCDHELLOABC";  
var I = S1.length;
```

substring – returns the substring of string using a starting and ending point.

will put the number 11 into variable I.

```
/* 01234567789 */  
var a = "Hello World";  
var b = a.substring(4,8);
```

substr – returns the substring of a string using a starting position and a length

will put "o Wor" into b.

```
/* 01234567789 */  
var a = "Hello World";  
var b = a.substr(2,3);
```

toLowerCase – returns the lowercase of string

will put "llo" into b.

```
var a = "Hello World";  
var b = a.toLowerCase();
```

toUpperCase – returns the uppercase of a string

will put "hello world" into b.

```
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";
```

JavaScript Coding Styles

In coding RAMP-TS scripts in JavaScript these code fragments are all standard and equivalent:

```
x = new Object();  
x.a = 1;  
x.b = "Hello";
```

```
x = { a : 1, b : "Hello" }
```

```
x = { "a" : 1, "b" : "hello" } <===== which is the JavaScript  
format that was chosen for use in AJAX-JSON strings.
```

```
x = { };  
x["a"] = 1;  
x["b"] = "hello";
```

```
x = new Object();  
x["a"] = 1;  
x["b"] = "hello";
```

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 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_SCREEN("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 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](#)

[Scripting Pop-up Menu](#)

[Updating the Instance List from RAMP screens](#)

[Subfiles/Browselists](#)

[Script Skeletons](#)

[Script Functions](#)

[Framework Objects that Scripts Can Refer To](#)

[User-Defined Script Functions](#)

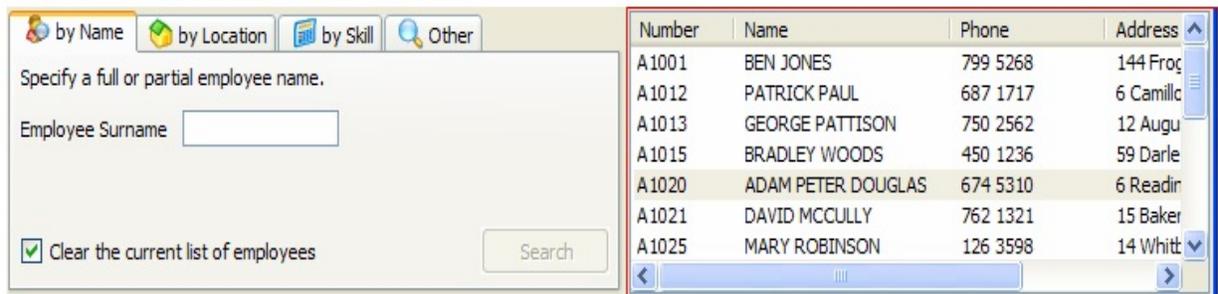
[When Are Scripts Reloaded so That Change Can Be Tested?](#)

[Switching Off Recursion Checking](#)

[Advanced Scripting](#)

Interacting with Instance Lists in Scripts

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):



The screenshot shows a search interface for employee instances. On the left, there are four tabs: "by Name" (selected), "by Location", "by Skill", and "Other". Below the tabs, there is a text input field labeled "Employee Surname" and a "Search" button. A checkbox labeled "Clear the current list of employees" is checked. On the right, a table displays the search results. The table has four columns: "Number", "Name", "Phone", and "Address". The table is outlined in red. The data in the table is as follows:

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

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 a 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:

```
/* 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);
```

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 LANSAs 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(#Department) 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:

Microsoft Internet Explorer



Current employee number is A1020

OK

Microsoft Internet Explorer



Current employee name is ADAM PETER DOUGLAS

OK

Working with All Selected Entries

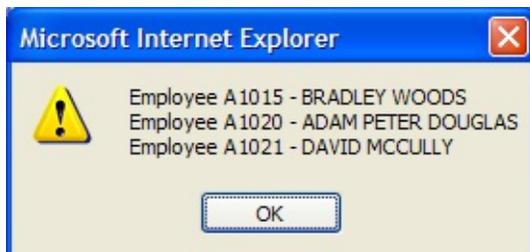
More than one entry can be selected in the instance list. This script displays the number and name of all selected employees in a message:

```
/* 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);  
}
```

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

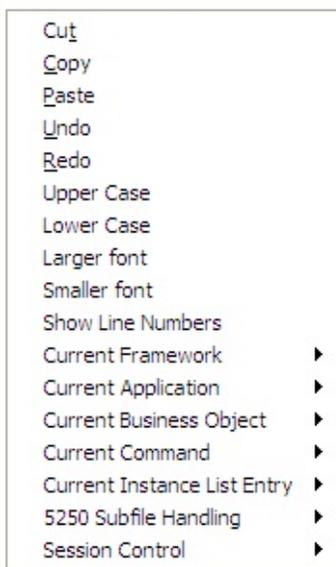
Number	Name	Phone	Address
A1001	BEN JONES	799 5268	144 Frog
A1012	PATRICK PAUL	687 1717	6 Camille
A1013	GEORGE PATTISON	750 2562	12 August
A1015	BRADLEY WOODS	450 1236	59 Darle
A1020	ADAM PETER DOUGLAS	674 5310	6 Reading
A1021	DAVID MCCULLY	762 1321	15 Baker
A1025	MARY ROBINSON	126 3598	14 Whit

It would display this alert message:



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 Area](#).



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](#)

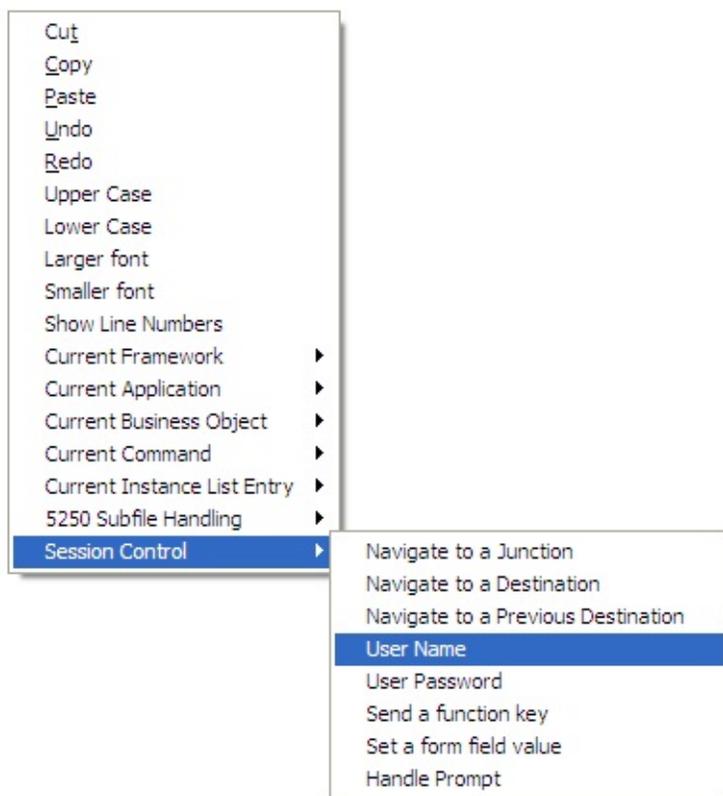
[Adding Your Own Options to the Scripting Pop-Up Menu](#)

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);
```

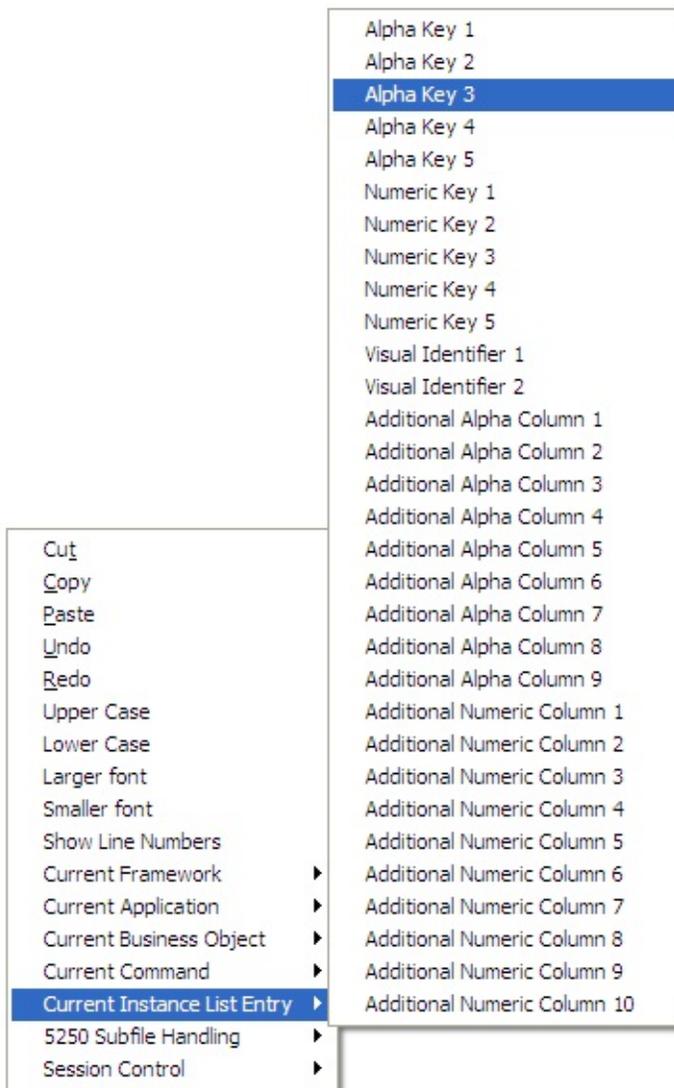
Replacing Hardcoded Employee Number with Current Instance List Entry

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

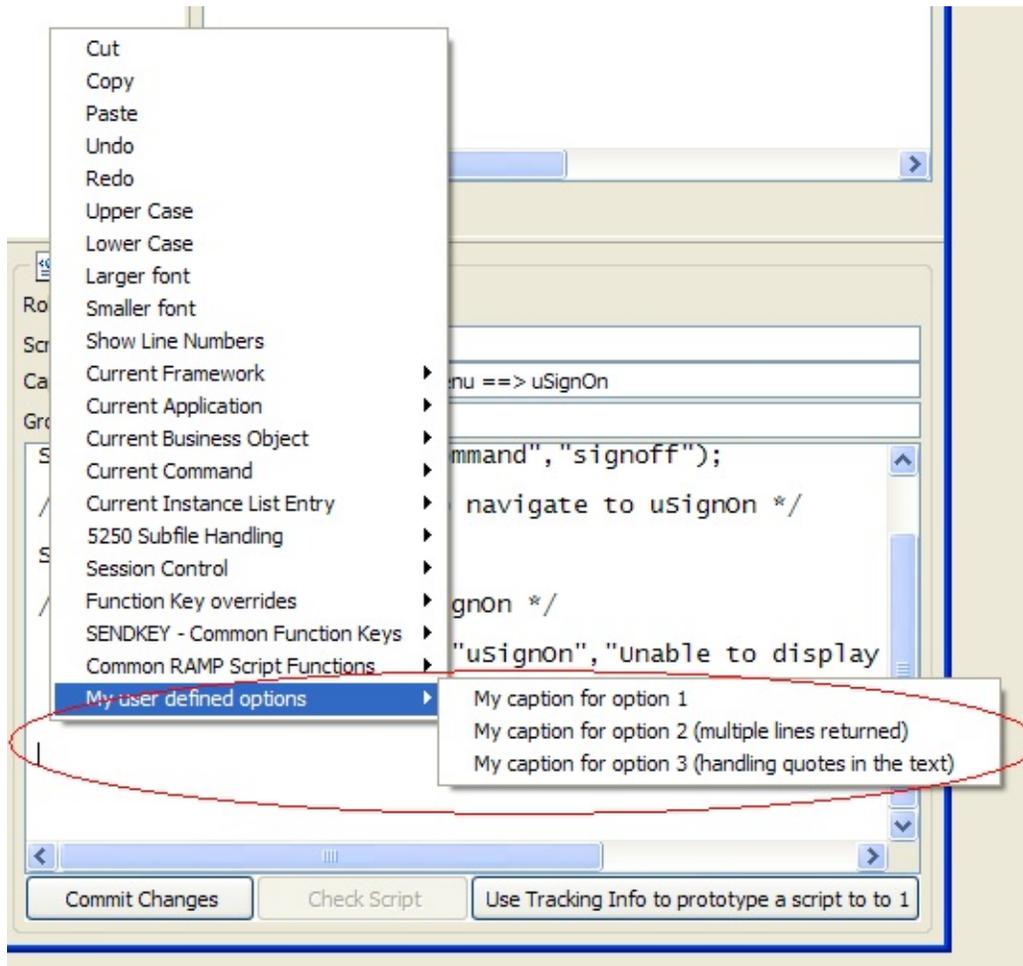
You can add your own options to the scripting pop up menu by creating an xml file called uf_um835.xml, and putting it in the partition execute directory. You can do this using notepad.

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 - &quot;" />
  <PROPERTY NAME="STRING" VALUE="Less than - &lt;" />
  <PROPERTY NAME="STRING" VALUE="Greater than - &gt;" />
</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.

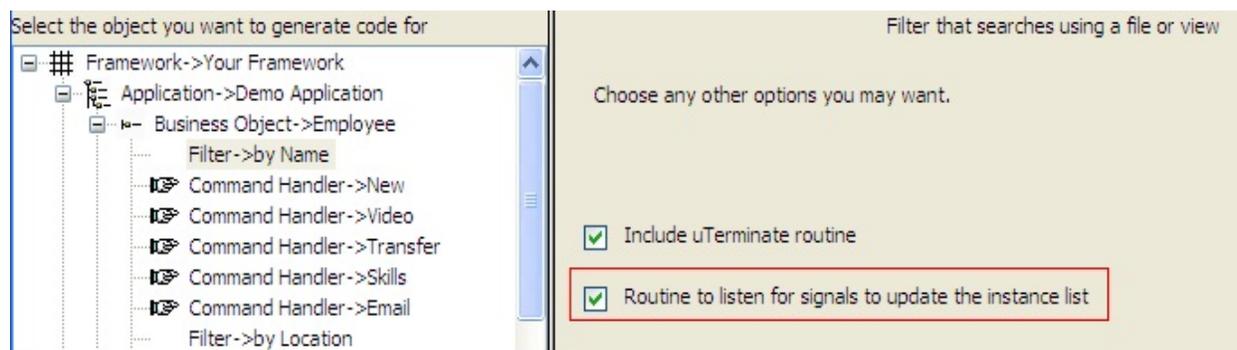
Updating the Instance List from RAMP screens

The tutorial [RAMP-TS007: Snapping in a Data Entry Function](#) 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 #DEPARTMENT #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(#DEPARTMENT) 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(#DEPARTMENT) 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(#DEPARTMENT)
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. Here are some techniques may be useful to you in different situations:

[Subfile Lines per Entry](#)

[Identifying Subfile fields](#)

[Referencing Subfile fields](#)

[Iterating Subfile Rows](#)

[Paging down or up a subfile](#)

[Locating a specific value in a Subfile and making the selection](#)

Subfile Lines per Entry

When a Subfile uses more than 1 line per entry you must set the appropriate value in this setting:



Start	End	Name	Lines/Entry
18	21	R@F00003	1

If not, the result of the SETVALUEs and GETVALUEs of Subfile fields will be incorrect.

Identifying Subfile fields

There is no difference between naming Subfile fields and naming any other fields. Conceptually though, when you are setting the name of a field in a Subfile you have to think that you are actually setting the name of a Subfile column.

In this picture the focus is on the Selection field. Notice how the entire column is highlighted in the 5250 screen:

Focus on Sel field column

All fields in column selected

<input type="checkbox"/>		4	2
<input type="checkbox"/>	Sel		
<input type="checkbox"/>		4	11
<input type="checkbox"/>	Code		
<input type="checkbox"/>		4	20
<input type="checkbox"/>	Description		
<input type="checkbox"/>	SFL_SELECT	5	5 Y
<input type="checkbox"/>		5	13 Y
<input type="checkbox"/>	ADM		
<input type="checkbox"/>	SFL_DEPTDESC	5	20 Y
<input type="checkbox"/>	ADMINISTRATOR DEPT		
<input type="checkbox"/>		22	2
Select department to review/change/delete			

Sel	Dept Code	Department Description
<input type="checkbox"/>	ADM	ADMINISTRATOR DEPT
<input type="checkbox"/>	AUD	INTERNAL AUDITING
<input type="checkbox"/>	FLT	FLEET ADMINISTRATION
<input type="checkbox"/>	GAC	GROUP ACCOUNTS
<input type="checkbox"/>	INF	INFORMATION SERVICES
<input type="checkbox"/>	LEG	LEGAL DEPARTMENT
<input type="checkbox"/>	MIS	MANAGEMENT INFORMATION
<input type="checkbox"/>	MKT	MARKETING DEPARTMENT
<input type="checkbox"/>	SD	SALES & DISTRIBUTION
<input type="checkbox"/>	TRVL	TRAVEL DEPARTMENT
<input type="checkbox"/>	TST	TESTING

Referencing Subfile fields

To reference a field in a Subfile the field name by itself is not enough because the name only resolves to a column in the Subfile. To reference a field in a column and row you must also specify a row Index. The index is an integer starting from 1.

For example, to set the value of the selector field in the third row to "X":

```
SETVALUE("SFL_SELECT","X", 3);
```

See [SETVALUE Function](#).

To get the department description in the seventh row (note we named this field as SFL_DEPTDESC):

```
var sDeptDesc = GETVALUE("SFL_DEPTDESC", 7);
```

See [GETVALUE Function](#).

Iterating Subfile Rows

To iterate subfile rows you need to know when you have reached the last row in order to stop your logic. To do this use the [CHECK_FIELD_EXISTS Function](#):

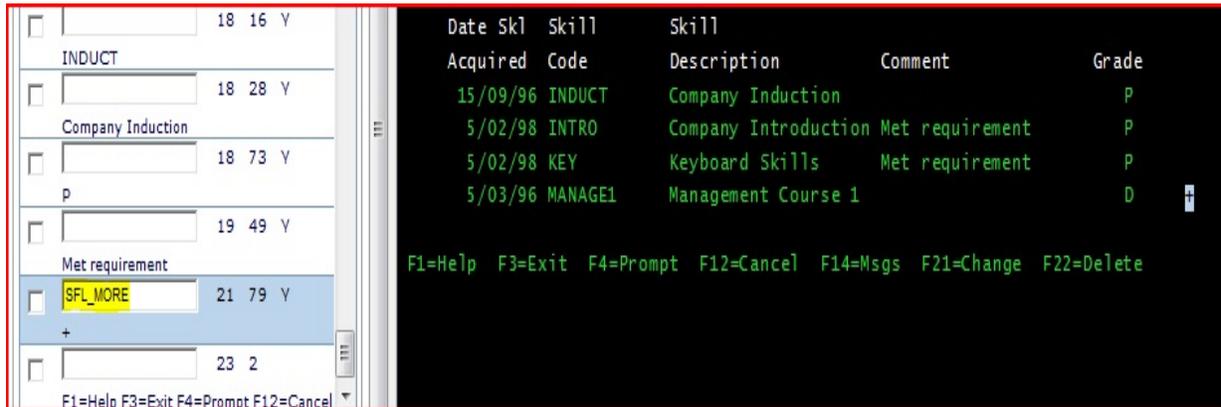
```
while ( CHECK_FIELD_EXISTS(sFindField,iInd) )
{
    <your logic>
}
```

where sFindField is any of the named subfile fields.

Paging down or up a subfile

To make the Subfile page to the next or previous screen you should set a name to the Subfile indicator that tells you whether there is another page to show.

This is usually a plus sign ("+") but may vary.



If the field is present we can assume there is another page:

```
if ( CHECK_FIELD_EXISTS("SFL_MORE") )
{
    SENDKEY(KeyPageDown);
}
```

Locating a specific value in a Subfile and making the selection

Case A: when a selection is found, set the cursor on the appropriate row and press Enter.

```
uSubfileSearch: function(sToForm, sFindValue, sFindField, sMoreRecsField)
{
    /* Subfile indexed fields are one based */

    var iInd = 1;
    var bFound = false;

    while( (CHECK_FIELD_EXISTS(sFindField,iInd)) && !(bFound) )
    {
        /* Found, set the flag to true to cause the loop to end */
        if (sFindValue == GETVALUE(sFindField,iInd))
        {
            bFound = true;
        }
        else /* Increase field index */
        {
            iInd++;
        }
    }

    /* If found, position the cursor to the field and index and send an Enter
    key to cause that entry to be selected */
    if (bFound)
    {
        SETCURSORTOFIELD(sFindField,iInd);
        SENDKEY(KeyEnter);
        Q_CHECK_CURRENT_FORM(sToForm,"Unable to navigate to " +
sToForm);
    }

    /* If not found, check whether the nominated more records indicator field
    is present on the screen. If it is we can page down. */
    /* Note the payload accompanying the Sendkey. It is used in the
    vHandleArrive function to decide whether we have to repeat this logic. */
}
```

```
else if ( CHECK_FIELD_EXISTS(sMoreRecsField) )
{
    SENDKEY(KeyPageDown, "Next_Page");
}
}
```

Case B: when a selection is found, set the cursor on the appropriate row, set the value of the selector field "SFL_SELECT" to "2" and press Enter.

This case is almost the same as the prior one except for the SETCURSOR call. Replace the SETCURSOR with

```
SETVALUE("SFL_SELECT", "2", iInd);
```

Script Skeletons

Scripts for Destinations, Junctions and Special are created based on specific skeleton files located in your partition execute folder. These files are called:

vf_fpm030_D.dat – skeleton for Destinations

vf_fpm030_J.dat – skeleton for Junctions

vf_fpm030_S.dat – skeleton for Specials

vf_fpm030_U.dat – skeleton for Unknowns

The skeletons lay out the basic script sections but also include some default behaviour.

You can change the skeletons if the default behaviour doesn't accommodate your specific needs. Edit the skeleton files with any text editor like Notepad.

For example, the skeleton for a Destination has this line:

```
GET_FORM_MESSAGE(22)
```

Because most 5250 applications use line 22 to send their messages. If you application sends messages to a different line you may want to change the skeleton.

Note that the skeleton is only used when a script is created. Skeleton changes will have no effect on existing scripts.

Warning: Product upgrades or reinstallations will overwrite these skeleton files. You must reapply any changes you make after each upgrade or reinstallation.

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!

ADD_STRING Function	Defines a string by a unique number for use by other scripts
ADD_UNKNOWN_FORM_GUESS Function	Function keys to send when an unknown form appears during RAMP navigation
ALERT_MESSAGE Function	Issue a message as an alert
AVCLOSEFORM Function	Signal to the Framework to close the current screen
AVRESTOREAVALUE and AVRESTORENVALUE Function	Restores an alphanumeric or numeric value from the Framework virtual clipboard
AVSAVEVALUE Function	Save an alphanumeric or numeric value in the Framework virtual clipboard.
AVSIGNALEVENT Function	Signal an event to filters and RAMP screens
AVSWITCH Function	Requests a switch to another business object and optionally the execution of a nominated command.
CHECK_CURRENT_FORM Function	Check that RAMP is showing a screen
CHECK_FIELD_EXISTS Function	Checks whether a field is present in the current screen
CLEAR_MESSAGES Function	Clear all messages currently in the stack

COPYTOCLIPBOARD Function	Copy a string to the user's clipboard
CURRENT_FORM Function	Get the Form Name of the current RAMP screen
DROP_SPECIAL_FIELD_HANDLER Function	Removes the definition of a 5250 special field handler
FATAL_MESSAGE Function	Issue a fatal message
FATAL_MESSAGE_TYPE Function	Stop the Framework from shutting down when a fatal navigation error occurs
GET_FORM_MESSAGE Function	Get the 5250 message text at a specified row number and optionally route as a Framework message.
GETVALUE Function	Get the value from a field on a RAMP screen
HANDLE_PROMPT Function	Show a user defined prompter form for a field
HIDE_5250_BUTTONS() Function	Use in the ARRIVE script to hide the function key buttons in the 5250 screen
HIDE_CURRENT_FORM Function	Hide the current screen with an optional message
LOCK_FRAMEWORK Function	Locks the framework
MESSAGE Function	Issue a message
NAVIGATE_TO_SCREEN Function	Navigate to a screen
OVERRIDE_KEY_CAPTION_ALL Function	Assigns a new caption for a function key on any screen
OVERRIDE_KEY_CAPTION_SCREEN Function	Assigns a new caption for a function key on a particular screen

OVRSF LAUTOGUI Function	Allows you to turn the Axes system flag Recognise subfiles as tables on and off on a screen by screen basis.
Q_CHECK_CURRENT_FORM Function	Check that RAMP is showing a screen, the request is queued up and processed when the next screen arrives
Q_NAVIGATE_TO_SCREEN Function	Navigates to a nominated 5250 screen, the request is queued up and processed when the next screen arrives
Q_SENDKEY Function	Emulates the pressing of a key, the request is queued up and processed when the next screen arrives
Q_SETVALUE Function	Set the content of a field on a 5250 screen to a value, the request is queued up and processed when the next screen arrives
RESTART_LAST_NAVIGATION Function	Restarts to the last navigation plan when the next screen arrives.
SCREEN Function	Returns the screen object for a specified screen name
SENDKEY Function	Emulate pressing a key.
SET_HANDLER_CAPTION Function	Set the current command handler caption to a new value
SET_SPECIAL_FIELD_HANDLER Function	Sets or resets the current definition of a 5250 special field handler
SETBUSY Function	Moves the cursor to the specified field, or piece of text, or subfile cell
SETCURSORTOFIELD Function	Moves the cursor to the specified field, or piece of text or subfile cell

SETCURSOROROW Function	Moves the cursor to the specified row and column on the screen
SETKEYENABLED Function	Dynamically enable or disable a Destination's button or 5250 function key
SETVALUE Function	Set a field on a RAMP screen to a value
SHOW_5250_BUTTONS() Function	Use in the ARRIVE script to show the function key buttons in the 5250 screen
SHOW_CURRENT_FORM Function	Show or hide the current screen.
SHOWSTATUSBAR Function	Show/Hide the 5250 terminal status bar for the end user
STRING Function	Returns the string for a given string identification number
STRIP_LEADING_NUMBERS Function	Returns the leading numbers from a string
TONUMBER Function	Makes a string or other JScript object into a number.
TOSTRING Function	Makes a number or null or other JScript object into a string.
TRACE Function	Add run time information to the trace panel
TRIM_LEFT Function	Trim preceding (left) spaces from a string
TRIM_RIGHT Function	Trim trailing spaces from a string
UNLOCK_FRAMEWORK Function	Unlocks the framework

LOCK_FRAMEWORK Function

Locks the framework and specifies the locking message to be displayed.

Equivalent to using Set #avFrameworkManager

uLocked(USER|PROGRAM|PROGRAM_EXIT) uLockedMessage('message')
in a VL component.

Only valid in VLF-WIN environments – ignored in other environments.

Syntax

LOCK_FRAMEWORK(lock type, lock message)

Parameters

Lock type Required. A string containing the type of lock required as "USER", "PROGRAM" or "PROGRAM_EXIT".

USER means that the Framework is locked, but that the user can elect to end the lock.

PROGRAM means that the Framework is locked and only a program can unlock it.

PROGRAM_EXIT means that the Framework is locked except when exiting or closing down and only a program can unlock it.

Lock message Required. A string containing the message to be shown to the user if they attempt to do something that would violate the lock state.

Return Value

None

Example

```
LOCK_FRAMEWORK("PROGRAM", "You need to return to the details screen before attempting this action");
```

UNLOCK_FRAMEWORK Function

Unlocks the framework and clears the current locking message.

Equivalent to using Set #avFrameworkManager uLocked(FALSE)
uLockedMessage(" ") in a VL component.

Only valid in VLF-WIN environments – ignored in other environments.

Syntax

```
UNLOCK_FRAMEWORK()
```

Parameters

None.

Return Value

None.

Example

```
UNLOCK_FRAMEWORK();
```

RESTART_LAST_NAVIGATION Function

Restarts to the last navigation plan when the next screen arrives.

Syntax

```
RESTART_LAST_NAVIGATION()
```

Parameters

None.

Return Value

"" (null string) or the last function key used.

Remarks

This function is used to restart the last navigation performed or to resend the last key.

Typically it is used in the arrival script of an unknown or special screen that needs to be logically eliminated from the screen flow during a navigation – for example a break message screen.

A call to this function needs to be followed by a SENDKEY() function call to cause the current screen to be removed from the screen flow.

When the break message screen appears while navigating between two screens, the function should return "" (null string). When the next screen arrives the navigation plan that was in progress is restarted and replanned to resume the screen flow.

If it appears during a user initiated action like pressing a function key or button, the function returns the last function key that was used before the break message screen appeared. This allows the user to re-send the last key to continue the screen flow.

Example

```
var lastFkey = RESTART_LAST_NAVIGATION();  
SENDKEY(KeyF3);  
if (lastFkey != "") Q_SENDKEY("",lastFkey); /* queue sending of the last  
function key before the break message screen appeared */
```

Note that this function should only be invoked from an arrival script and would

almost always need to be immediately followed by a `SENDKEY()` function call to cause the current screen to be removed.

OVRSFLLAUTOGUI Function

Allows you to turn the Axes system flag Recognise subfiles as tables on and off on a screen by screen basis.

You need to be on aXes 2.1 or later to use this scripting function.

Syntax

OVRSFLLAUTOGUI(sScreenName, bOvr)

Parameters

sScreenName Required. An string that specifies the name of the screen.

bOvr Optional. Boolean.

true will cause the subfiles to be recognised as.

false will turn the setting off

For any other value including no value, the behaviour will be according to the current subfile AutoGUI setting.

Remarks

The only valid place to put a call to this API is in the Navigate_TO section of the **Login script** for example:

```
vHandle_NAVIGATETO: function(sToForm, oPreviousForm)
{
    var bReturn = true;
    HIDE_CURRENT_FORM();
    SETBUSY(true);
    OVRSFLLAUTOGUI("Employee_Skills", false);
    etc
}
```

AVSWITCH Function

Requests a switch to another business object and optionally the execution of a nominated command.

Syntax

AVSAVEVALUE(sTo, sNamed, sExecute, sTargetWindow)

Parameters

sTo	Specify as FRAMEWORK, APPLICATION or BUSINESSOBJECT indicating the object to which control is to be switched.
sNamed	Specifies the User Object Name/Type of the APPLICATION or BUSINESSOBJECT that control is to switch to.
sExecute	Specify the User Object Name/Type of any command that is to be executed in the target FRAMEWORK, APPLICATION or BUSINESSOBJECT.
sTargetWindow	Specifies the target window in which the switch operation should be performed. Allowable values are CURRENT (the current window), MAIN (the main window) or specific window name. The default value is MAIN.

Return Value

None.

Remarks

It is the function callers responsibility to ensure the RAMP-TS session is not busy when the call to AVSWITCH is made. In development mode this will result in the RAMP Session Busy message.

Examples

```
/* Switch to the Combobox Fast Part examples and run Example 1 */
```

```
AVSWITCH("BUSINESSOBJECT","C0846821929747C295C29FF1E518CC.
```

```
/* Switch to a business object in the same additional window and */
```

```
/* run the details command. */
```

```
AVSWITCH("BUSINESSOBJECT","EMPLOYEES","DETAILS","CURREN
```

TRIM_RIGHT Function

Trim trailing spaces from a string.

Syntax

```
var sTrimmed = TRIM_RIGHT(sString);
```

Parameters

sString Required. The string to be right trimmed.

Return Value

String. Returns the right trimmed string.

Example

```
var myString = GETVALUE("PageMarker");  
var sTrimmed = TRIM_RIGHT(myString);
```

TRIM_LEFT Function

Trim preceding (left) spaces from a string.

Syntax

```
var sTrimmed = TRIM_LEFT(sString);
```

Parameters

sString Required. The string to be left trimmed.

Return Value

String. Returns the left trimmed string.

Example:

```
var myString = GETVALUE("PageMarker");  
var sTrimmed = TRIM_LEFT(myString);
```

SHOW_CURRENT_FORM Function

Show or hide the current screen.

Syntax

SHOW_CURRENT_FORM(bShow)

Parameters

bShow Required. A boolean value that indicates whether to show the current screen.

Return Value

None

Example

Show the current screen (in an arrival script):

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;
    SHOW_CURRENT_FORM(true);
}
```

SHOWSTATUSBAR Function

Show/Hide the 5250 terminal status bar for the end user.

Syntax

SHOWSTATUSBAR(bShow)

Parameters

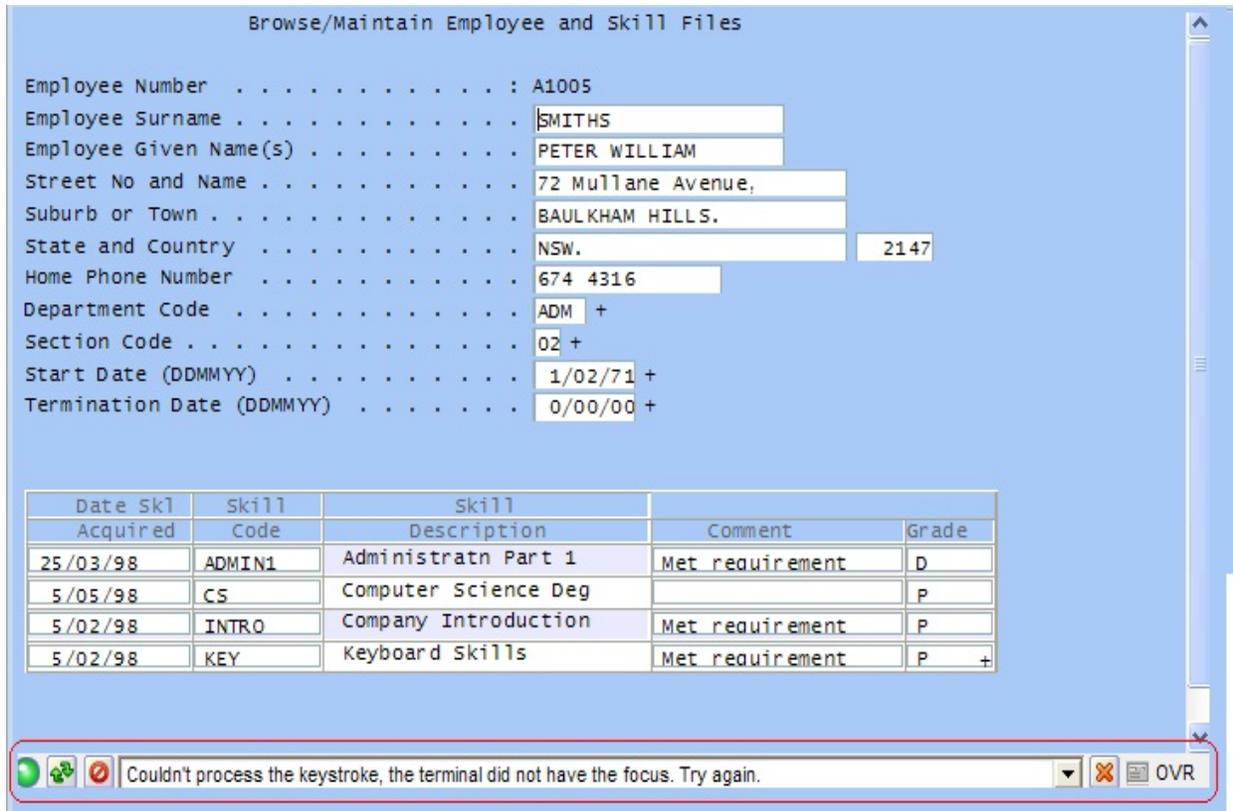
bShow Required. A boolean value that indicates whether to show the 5250 terminal status bar.

Return Value

None

Remarks

The 5250 terminal status bar looks like this (in red)



It allows the end user to carry out a variety of functions, such as stopping or refreshing the page and displaying messages from the terminal

Example

Show the end user the 5250 terminal status bar (in an arrival script):

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
  var bReturn = true;
  SHOW_CURRENT_FORM(true);
  HIDE_5250_BUTTONS();
  SHOWSTATUSBAR(true);
}
```

TOSTRING Function

Makes a number or null or other JScript object into a string.

Syntax

```
myString = TOSTRING(oObject);
```

Parameters

oObject	Required. A JScript object. Usually it would be a number, that needs to be converted to a string. It can also be a string or other JScript object.
---------	--

Return Value

String	The returned string
--------	---------------------

Example

Convert the number 12.5 into a string

```
var myString = TOSTRING(12.5);
```

TONUMBER Function

Makes a string or other JScript object into a number.

Syntax

```
myNumber = TONUMBER(oObject);
```

Parameters

oObject	Required. A JScript object. Usually it would be a string, that needs to be converted to a number. It can also be a number.
---------	--

Return Value

Number	Required. The returned number
--------	-------------------------------

Example

Convert the string "12.5" into a number

```
var myNumber = TONUMBER("12.5");
```

SETBUSY Function

Enables/Disables the system busy state. When SETBUSY is set to true, the status light will change to red and all user interactions are ignored.

Syntax

SETBUSY(fState)

Parameters

fState Required. A boolean value that indicates whether the system should go into the busy state.

Return Value

None

Remarks

Setting SETBUSY(true) indicates that:

The system is busy doing something

Things the user does should be ignored while the system is busy.

There is nothing the user can or should do to release the busy state - it will be released automatically when the busy activity completes.

It should not be enabled when interaction is required from the user.

If a script sets busy to true, it should ensure that it is set to false after the processing is finished. Otherwise the user will not be able to interact with the aXes screen.

Example

Turn off the busy state to allow user interaction, and indicate that the system is not busy

```
SETBUSY(false);
```

Turn on the busy state to ignore user interaction, and indicate that the system is busy

```
SETBUSY(true);
```

CHECK_FIELD_EXISTS Function

Checks whether a field is present in the current screen. Use the optional iInd parameter to refer to the instance of a field in a subfile.

Syntax

CHECK_FIELD_EXISTS(sFieldName, [iInd])

Parameters

sFieldName	Required. A string that contains the name of the field to check.
iInd	Optional. Integer, must be greater than zero. For subfile fields this is the specific instance of the field.

Return Value

Boolean. Returns one of the following possible values:

true	The field was found in the current screen.
false	The field was not found in the current screen.

Remarks

For subfile fields you may choose not to pass iInd. When passed, iInd must be greater than zero or else it will return false.

Example

Check that that the field named PageMarker is present on this screen:

```
if (CHECK_FIELD_EXISTS("PageMarker"))
```

Read every subfile entry on the page

```
for ( Index = 1; CHECK_FIELD_EXISTS("ColGivename", Index); Index++ )
{
  var Givename = GETVALUE("ColGivename",Index);
  var Empno = GETVALUE("ColEmpno", Index);
  ...
}
```

DROP_SPECIAL_FIELD_HANDLER Function

Removes the definition of a 5250 special field handler. You should complete the [RAMP-TSAD03: Special Field Handling](#) tutorial before using this function.

If the handler definition does not exist the request is ignored and no error results. Removing the definition of a displayed handler will not impact it until it needs to be displayed again.

Syntax

```
DROP_SPECIAL_FIELD_HANDLER(sName,sKey)
```

Parameters

sName Required. The 5250 name of the field associated with this special field handler.

sKey Required. The function key that causes the handler to be invoked. Use the same keys names as used in SENDKEY() operations.

Return Value

None

Examples

See the [RAMP-TSAD03: Special Field Handling](#) tutorial for examples.

SET_SPECIAL_FIELD_HANDLER Function

Sets or resets the current definition of a 5250 special field handler. You should complete the [RAMP-TSAD03: Special Field Handling](#) tutorial before using this function.

Syntax

SET_SPECIAL_FIELD_HANDLER(sName,sKey,sHandler,sInfo1,sInfo2,sInfo3)

Parameters

- `_sName` Required. The 5250 name of the field associated with this special field handler.
- `sKey` Required. The function key that causes the handler to be invoked. Use the same keys names as used in SENDKEY() operations.
- `sHandler` Required. The name of the VL component special field handler. The VL component must be a class VF_AC017 object.
- `sInfo1 – sInfo3` Optional. Additional string information to be passed to the handler if it is later invoked.

Return Value

None

Examples

See the [RAMP-TSAD03: Special Field Handling](#) tutorial for examples.

GET_FORM_MESSAGE Function

Get the 5250 message text at a specified row number and optionally route as a Framework message.

Syntax

```
var bMoreRecords = GET_FORM_MESSAGE([iRow,] [sMoreIndicator,]  
[bRoute])
```

Parameters

iRow	Optional. An integer specifying the message row number. Defaults to the last row.
sMoreIndicator	Optional. The string used by the Application to denote whether there are more messages available. Defaults to "+".
bRoute	Optional. A Boolean to specify whether the message is to be routed to the Framework message area. When true, the text of the message in the screen will be removed. Defaults to true.

Return Value

Boolean. Returns one of the following possible values:

true	The more indicator was found in an element displayed on the message line
false	The more indicator was not found in an element displayed on the message line

Remarks

Additionally, RAMP will set a property called vLastMessage in the current form

object that will contain the text of the last message retrieved. To use this property in your script use:

```
var sLastMsg = this.vLastMessage;
```

Note that the use of the this pointer is only valid within the current script.

Examples

```
GET_FORM_MESSAGE(22); /* Extract messages and hide the message line  
if (this.vLastMessage != "") ALERT_MESSAGE("ERROR:" + this.vLastMess  
GET_FORM_MESSAGE(22, "More");
```

The following example shows using GET_FORM_MESSAGES in an Arrival Script to rout all 5250 messages to the Framework

The GET_FORM_MESSAGE retrieves the text visible on the 5250 screen at the nominated line.

If the 5250 screen indicates that there are more messages available the function will return a result of true. For the other messages to be read they must be made visible. This is achieved by setting the cursor to the line displaying the message and sending a Page Down key to the 5250 screen. When the 5250 screen arrives the new message is retrieved. Note that this is an expensive exercise.

```
/* =====  
/* ===== vHandle_ARRIVE ===== */  
/* =====  
/* Handle arrival at this Destination */  
/* oPayload: The payload supplied by the event initiator */  
/* oPreviousForm: Reference to previous object Form*/  
  
vHandle_ARRIVE: function(oPayload, oPreviousForm)  
{  
  var bReturn = true;  
  
  SHOW_CURRENT_FORM(true);  
  HIDE_5250_BUTTONS();  
  SETBUSY(false); /* Turn off the busy state to allow user interaction */
```

```
/* Get the 5250 message text from the message area */
var flagMoreRecords = GET_FORM_MESSAGE(22);

/* If there are more messages */
if (flagMoreRecords == true)
{

    /* Move the cursor to the line displaying the Messages */
    SETCURSORTOROW(22);

    /* Bring up the next message */
    SENDKEY(KeyPageDown);

}

/* <ARRIVE /> - Do not remove or alter this line */

return(bReturn);
},
```

SCREEN Function

Returns the screen object for a specified screen name.

Syntax

```
var oScreen = SCREEN(sName)
```

Parameters

sName	Required. String that specifies the name of a screen.
-------	---

Return Value

oScreen	Object. A reference to a screen object.
---------	---

Remarks

Useful to access the properties of a specific screen at any point in time during your navigation.

Examples

See [Using Screen References](#) .

HIDE_5250_BUTTONS() Function

Use in the ARRIVE script to hide the function key buttons in the 5250 screen.

.

Syntax

```
HIDE_5250_BUTTONS();
```

Parameters

None

Example

```
HIDE_5250_BUTTONS();
```

SHOW_5250_BUTTONS() Function

Use in the ARRIVE script to show the function key buttons in the 5250 screen.

.

Syntax

```
SHOW_5250_BUTTONS();
```

Parameters

None

Example

```
SHOW_5250_BUTTONS();
```

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 the 5250 screen and show the error.

INFO – the error message will be routed to the Framework message area.

Return Value

None

Example

```
FATAL_MESSAGE_TYPE("HIDE");
```

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

Remarks

To be immediately effective, SETKEYENABLED needs to occur prior to SHOW_CURRENT_FORM(true) in an arrival script:

```
SETKEYENABLED("DisplayEmployee",KeyF5, false, false);  
SHOW_CURRENT_FORM(true);
```

Example

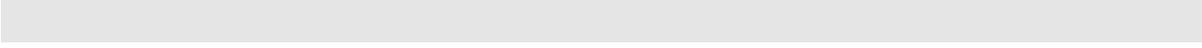
The Destination named uDisplayEmployee was set up to Show the prompt button but disable the F4 5250 function key.

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, iIndex)

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.

iIndex Optional. An Integer that specifies the subfile row of the field, for fields that are part of a subfile.

Note: the specified row index must exist in the current subfile page. CHECK_FIELD_EXISTS can be used to check whether a particular row exists in the subfile.

Return Value

None

Remarks

To set a value of a field on a screen by name, the field must have been given a name.

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");
```

Q_SETVALUE Function

Set the content of a field on a 5250 screen to a value, the request is queued up and processed when the next screen arrives. The field may be identified by name or by its order on the screen.

Syntax

```
function Q_SETVALUE(argCondition, sVariable, sValue, iIndex)
```

Parameters

argCondition Optional. May be passed as:

- null or "", indicating that no condition applies
- "=<<screen name when the queued instruction is executed>>" indicating that the resulting SENDKEY/SETVALUE should only be performed if the next screen is as named.
- "!=<<screen name when the queued instruction is executed>>" indicating that the resulting SENDKEY/SETVALUE should only be performed if the next screen is not as named.

The screen name condition is tested when the next or resulting screen arrives. The name specified does not have anything to do with the current screen name.

sVariable Required. String that contains the RAMP field name.

sValue Required. String that contains the value to set the field to.

iIndex Optional. An Integer that specifies the subfile row of the field, for fields that are part of a subfile.

Note: the specified row index must exist in the current subfile page. CHECK_FIELD_EXISTS can be used to check whether a particular row exists in the subfile.

GETVALUE Function

Get the value from a field on a RAMP screen.

Syntax

GETVALUE(sVariable)

Parameters

sVariable Required. String that contains the field name.

sIndex Optional. Specifies the row for fields used as columns in subfiles.

sIndex Optional. An Integer that specifies the subfile row of the field.
Note: the specified row index must exist in the current subfile page

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, oPayload)

Parameters

sKeyName Required. String that contains the name of the key. See [Function Key Names for SENDKEY Function](#).

oPayload Optional. Object that is passed with the function.

Return Value

None

Remarks

This function typically initiates an asynchronous 5250 server side operation. Your RAMP-TS script(s) should end all processing immediately after invoking this function and then do nothing more until the asynchronous operation completes.

The completion of the asynchronous operation is typically indicated by the execution of the arrival script of the resulting 5250 screen display. (Any queued script functions should be queued prior to executing this script function).

Examples

```
SENDKEY(KeyEnter);
```

The next example shows how to use the Payload Parameter with the SENDKEY and Q_SENDKEY functions.

An object is created and loaded with values in the Enter key BUTTONCLICK event and then the object is passed as the oPayload parameter of the SENDKEY function:

```

/* =====
/* =====  BUTTONCLICK  =====
/* =====
/* sButton: The button that was clicked */

vHandle_BUTTONCLICK: function(sButton)
{

    var bReturn = true;

    if (HANDLE_PROMPT()) return(bReturn); /* If the focus element is autom

/* <BUTTONCLICK /> - Do not remove or alter this line */

    /* Handle function keys and buttons */

    switch (sButton)
    {
    case KeyEnter:
        var objEmp = new Object();
        objEmp.strEmpno = GETVALUE("empno");
        objEmp.strGName = GETVALUE("givename");
        objEmp.strSName = GETVALUE("surname");
        SENDKEY(KeyEnter, objEmp);
        break;
    case KeyF3:
        SENDKEY(KeyF3);
        break;
    case KeyF4:
        SENDKEY(KeyF4);
        break;
    case KeyF12:
        SENDKEY(KeyF12);
        break;
    case KeyF14:
        SENDKEY(KeyF14);
        break;
    case KeyF21:

```

```

        SENDKEY(KeyF21);
        break;
    case KeyF22:
        SENDKEY(KeyF22);
        break;
    default:
        SENDKEY(sButton);
        break;
    }

    return(bReturn);
},

```

Then the vHandle_Arrive function of the resulting screen gets the values from the payload if one is passed:

```

/* =====
/* ===== vHandle_ARRIVE ===== */
/* =====
/* Handle arrival at this Destination */
/* oPayload: The payload supplied by the event initiator */
/* oPreviousForm: Reference to previous object Form*/

vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    var bReturn = true;

    SHOW_CURRENT_FORM(true); /* Show the form in the framework and
    HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons displayed
    GET_FORM_MESSAGE(22); /* Extract messages and hide the message
    SETBUSY(false); /* Last thing done - turn off the busy state */

    /* if there is something in the payload */
    if (oPayload != null)
    {
        ALERT_MESSAGE("Employee Details from the payload are: Employee
    }

```

```
/* <ARRIVE /> - Do not remove or alter this line */
```

```
return(bReturn);  
},
```

Q_SENDKEY Function

Emulates the pressing of a key, the request is queued up and processed when the next screen arrives.

Syntax

Q_SENDKEY(argCondition,sKeyName, oPayload)

Parameters

argCondition Optional. May be passed as:

- null or "", indicating that no condition applies
- "=<<screen name when the queued instruction is executed>>" indicating that the resulting SENDKEY/SETVALUE should only be performed if the next screen is as named.
- "!=<<screen name when the queued instruction is executed>>" indicating that the resulting SENDKEY/SETVALUE should only be performed if the next screen is not as named.

The screen name condition is tested when the next or resulting screen arrives. The name specified does not have anything to do with the current screen name.

sKeyToSend Required. String that contains the name of the key. See [Function Key Names for SENDKEY Function](#).

oPayload Optional. Object that is passed with the function.

Example

```
Q_SENDKEY("",KeyF12); /* Unconditionally send F12 against the resulting  
SENDKEY(KeyEnter); /* Send enter against the current screen */
```

Or

```
Q_SENDKEY("=Confirm",KeyEnter); /* If the resulting screen is named C  
Q_SENDKEY("!=Confirm",KeyF12); /* else send F12  
SENDKEY(KeyEnter); /* Send enter against the current screen */
```

Also see [SENDKEY Function](#).

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.

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;
```

Q_CHECK_CURRENT_FORM Function

Check that RAMP is showing a screen, the request is queued up and processed when the next screen arrives.

Syntax

```
Q_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.

Example

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 vHandle_NAVIGATETO function for this Junction or Destination. This will allow the Framework to navigate back to sign off and end the session cleanly.

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

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 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 Optional. Strings that are to be concatenated with the previous
-> N message text (a separator space is added).

Return Value

None

Notes

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

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() ;
```

SETCURSORTOROW Function

Moves the cursor to the specified row and column on the screen.

This function is intended to programmatically manipulate the coordinates (5250 row and column) of the underpinning 5250 screen cursor - rather than anything visual on the currently displayed web page. Typically a call to this function is immediately followed by a SENDKEY function call.

Syntax

SETCURSORTOROW(iRow, iColumn)

Parameters

iRow Required. Integer for the row on the screen. Starts at the top with row 1.

iColumn Optional. Integer, for the column on the screen. Must be greater than zero. Defaults to 1.

Return Value

None

Remarks

The cursor can be set to anywhere on the screen.

Example

Set the cursor to the 9th row down, 43rd column across

```
SETCURSORTOROW( 9 , 43);
```

SETCURSORTOFIELD Function

Moves the cursor to the specified field, or piece of text or subfile cell.

This function is intended to programmatically manipulate the coordinates (5250 row and column) of the underpinning 5250 screen cursor - rather than anything visual on the currently displayed web page. Typically a call to this function is immediately followed by a SENDKEY function call.

Syntax

SETCURSORTOFIELD(sFieldName, iInd)

Parameters

- sFieldName Required. A string that contains the name of the field to set the cursor to. This can be any named field or text or subfile column.
- iInd Required for subfiles. Integer, must be greater than zero. For subfile fields this is the specific instance of the field (the row within the column).

Return Value

None

Remarks

The cursor can be set to any named field or text or subfile column on the screen.

Example

Set the cursor to the field named as givenname, in RAMP-TS

```
SETCURSORTOFIELD("givenname");
```

Set the cursor to the field named ColDepartment in a subfile, fifth row down

```
SETCURSORTOFIELD("ColDeptment", 5);
```

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

Notes

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

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

Notes

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

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

Notes

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

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

Notes

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

The maximum length of the message strings when concatenated must be no more than 130 characters.

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

Notes

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

Example

```
TRACE("Inventory item", objListManager.AKey1[0] , "was deleted.");
```

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 may require. Note that by default the user defined prompter has bi-directional access to all named 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) ... returns "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_SCREEN Function

Navigates to a nominated 5250 screen.

Note that if you specify the name of the current destination the request will be ignored.

Syntax

NAVIGATE_TO_SCREEN(sScreenName, ForAction)

Parameters

SScreenName Required. A string that contains the name of a screen.

ForAction Char 256 – Optional.

String that contains a user defined value. This value will be returned to the Screen Wrapper in the vHandleArrive event.

Return Value

None

Remarks

This function typically initiates an asynchronous 5250 server side operation. Your RAMP-TS script(s) should end all processing immediately after invoking this function and then do nothing more until the asynchronous operation completes.

The completion of the asynchronous operation is typically indicated by the execution of the arrival script of the resulting 5250 screen display. (Any queued script functions should be queued prior to executing this script function).

Example

```
NAVIGATE_TO_SCREEN("Enrol Employee");
```

Q_NAVIGATE_TO_SCREEN Function

Navigates to a nominated 5250 screen, the request is queued up and processed when the next screen arrives.

Syntax

Q_NAVIGATE_TO_SCREEN(argCondition, sScreenName)

Parameters

argCondition Optional. May be passed as:

- null or "", indicating that no condition applies
- "=<<screen name when the queued instruction is executed>>" indicating that the resulting SENDKEY/SETVALUE should only be performed if the next screen is as named.
- "!=<<screen name when the queued instruction is executed>>" indicating that the resulting SENDKEY/SETVALUE should only be performed if the next screen is not as named.

The screen name condition is tested when the next or resulting screen arrives. The name specified does not have anything to do with the current screen name.

SscreenName Required. A string that contains the name of a screen.

Return Value

None

Example

```
Q_NAVIGATE_TO_SCREEN("=WorkwithCustomers","WorkwithCustomers_
```

Note that this function should only be invoked from a button script.

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_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 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 AVRESTOREAVALUE 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 AVRESTORENVALUE 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 | 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_SAVEVALUE](#) 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.

This request may be handled asynchronously. The consequences of invoking it may not be visible or useable until your currently active RAMP-TS script(s) complete executing and yields control back to the RAMP-TS manager.

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");
```

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. See [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);
```

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](#)

To find out how you can quickly enter these objects and their properties in your scripts, see [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	Type	Description
<<any property name>>	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*/
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	Type	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	Type	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	Type	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	Type	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 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	Type	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	Type	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	Type	Description
-----------------	-------------	--------------------

Name	string	The profile of the current User.
------	--------	----------------------------------

Password	string	The password of the current User.
----------	--------	-----------------------------------

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

See [RAMP-TSAD05: Using SHARED Properties and Functions](#) .

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.

When Are Scripts Reloaded so That Change Can Be Tested?

Destination, Special and Junction (Screen) scripts

They are reloaded into the execution environment when you click the commit button.

This means you can normally test script changes dynamically by just causing the modified script to re-execute (for example by clicking on another object in the instance list to re-execute an arrival script). You do not normally have to save and restart your VLF-RAMP sessions to test most modifications to screen scripts.

SHARED scripts in uf_sy420_rts.js

These scripts are loaded once when the VLF-RAMP session is opened.

Changing these scripts will have no impact until you open another VLF-RAMP session.

Tip: When testing SHARED scripts, open the business object you are using for testing them in an independent window.

When you change a SHARED script, save it to the server, close the independent window and then launch it again. This will close and reopen the VLF-RAMP session in the independent window, picking up the modified SHARED script.

Advanced Scripting

Creating your own navigation planner

Using Screen References

Using a vHandle_DEPART function

Arrival Scripting and Inter-Screen Communication

Creating your own navigation planner

When a framework users executes a command that is associated with a RAMP destination screen a plan is always made of how to navigate from the current 5250 screen to the required destination 5250 screen.

Normally this plan is constructed automatically by the RAMP framework. In specialized cases you can define your own navigation planning logic.

Imagine that destination screen DestinationA needs to take over the planning of how to best navigate to it.

To do this a function named `vHandle_USER_NAVIGATION_PLAN` is added to the script associated with screen DestinationA, like this:

```
vHandle_USER_NAVIGATION_PLAN: function()
{
    var bReturn = true;

    // your alternate navigation planning logic goes here

    return(bReturn); /* ⚡ Remember to return a Boolean success/fail value */
}, /* ⚡ Remember to separate this function from the others with a comma */
```

When the user executes a VLF command that is associated with DestinationA, instead of making its own plan, the RAMP framework will invoke function `vHandle_USER_NAVIGATION_PLAN` in DestinationA.

Typically the function `NAVIGATE_TO_SCREEN("screen name")` is used by this type of navigation planning function to initiate its own navigation plan.

Before using this feature, consider a screen named Destination1 that contains its own navigation planner coded like this

```
vHandle_USER_NAVIGATION_PLAN: function()
{
    NAVIGATE_TO_SCREEN("Destination1");
```

```
return(true);  
  
},
```

If you cannot see why this code could never possibly work you should not attempt to use this feature.

Using Screen References

The script associated with a screen definition in RAMP-TS defines a JavaScript object. In effect, the screen script is the object that defines the screen and what it can do.

By now you should have encountered the concept of adding properties to the definitions of your screens. Typically these are defined at the start of the script like this example:

```
{
  /* Properties of screen Destination1 */

  sCurrentOrder : "",
  fSkipIntroduction : false,
  fScrolling : true,
```

The various functions in this screens definition would reference them as `this.sCurrentOrder`, `this.sSkipIntroduction` and `this.fScrolling`.

They are useful for maintaining state within a screen definition and for communicating between different functions within the script.

By using the `SCREEN("Screen Name")` function you can obtain a reference to the named screens definition object.

For example, a junction Junction1 might have this code in its navigation script:

```
var oDest1 = SCREEN("Destination1");

oDest1.fSkipIntroduction = true;
oDest1.fScrolling = false;
```

This allows junction1 to directly access properties and even methods defined with destination screen Destination1.

This could have been coded:

```
SCREEN("Destination1").fSkipIntroduction = true;
```

```
SCREEN("Destination1").fScrolling
```

But this is not the best solution for two reasons:

SCREEN() will return a null reference if for some reason the definition of screen "Destination1" cannot be found, causing your script to fail.

The string "Destination1" needs to be converted to an object reference twice, so it is less efficient.

The most proper form of this code is therefore:

```
var oDest1 = SCREEN("Destination1");  
  
if (oDest1 != null)  
{  
    oDest1.fSkipIntroduction = true;  
    oDest1.fScrolling = false;  
}
```

Using a vHandle_DEPART function

You are probably familiar with screen scripts containing a vHandle_ARRIVE function that is executed when a 5250 screen arrives.

Any screen script can also, optionally, contain a vHandle_DEPART function. If it exists it will be invoked when a screen is being departed from, which is usually caused by a SENDKEY operation.

If you need to use one, add it to your screen's script like this example:

```
vHandle_DEPART: function()
{
    var bReturn = true;

    // your departure logic goes here

    return(bReturn); /* ⚠ Remember to return a Boolean success/fail value */
}, /* ⚠ Remember to separate this function from the others with a comma */
```

There are some things you should know about using vHandle_DEPART functions:

Using them is unusual. If you are using them a lot then it could be you are solving a problem using the wrong approach.

They are invoked when the current 5250 screen is submitted to server, usually as the result of SENDKEY operation. This means that they may activate when one of your other scripts is active (eg: the one that did the SENDKEY).

Using a SENDKEY operation in a departure script would be expected to fail or cause strange results. You should not do this.

They cannot cancel the submission of the screen. They are being informed that the screen is being submitted and allowed to perform any relevant close or cleanup actions. Returning false indicates that the vHandle_DEPART script failed. It does not indicate that the screen submission should be cancelled.

They are invoked every time a screen is submitted.

Arrival Scripting and Inter-Screen Communication

All screens have an arrival function.

Typically the arrival script of a screen defines a **single default behaviour**.

Typical single default behaviours are:

Junction screens – do nothing

Special screens - send key(s) to make the screen disappear

Destination screens – display the underlying 5250 screen

However, you will from time to time need to alter an arrival script in a junction, special or destination screen to make it support **multiple different behaviours**.

The most structured way of doing this is to first decide what behaviours you want your arrival script to support, and then to clearly define and document them at the start of you screen's script like this example:

```
RequestedArrivalBehaviour : 0,  
  
ArrivalBehaviours :  
{  
  Default      : 0, /* Default behaviour      */  
  SearchNext   : 1, /* Handle scroll up request */  
  SearchLast   : 2, /* Handle scroll down request */  
  ForcedNavigation : 3, /* Handle a forced navigation */  
  AutoConfirmation : 4 /* Handle auto confirmation */  
},
```

This very formally defines that this screen can support 5 different arrival behaviours.

Note 1: You do not have to define the behaviours this way and can use different names. This is an example of formal way to do this. This approach has documentation and debugging advantages. See the end

of this section form notes about using simpler approaches.

Note 2: These behaviours and their names are mythical examples. You can have as many behaviours as you like with any names.

Next, you actually need to change your screen's arrival script to handle these multiple behaviours. Again this can be done in a structured way, like this example arrival script for a destination screen:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
  /* Extract a copy of the requested behaviour */

  var RequestedBehaviour = this.RequestedArrivalBehaviour;

  /* Reset the requested behaviour back to the default behaviour */
  this.RequestedArrivalBehaviour = this.ArrivalBehaviours.Default;

  /* Now preform the requested behaviour */

  switch (RequestedBehaviour)
  {
    case this.ArrivalBehaviours.Default:
      SHOW_CURRENT_FORM(true);
      HIDE_5250_BUTTONS();
      SETBUSY(false);
      break;

    case this.ArrivalBehaviours.SearchNext:
      /* Logic to handle search next page behaviour*/
      break;

    case this.ArrivalBehaviours.SearchLast:
```

```

    /* Logic to handle search last page behaviour*/
    break;

    case this.ArrivalBehaviours.ForcedNavigation:
        /* Logic to handle a forced navigation, whatever that may be */
        break;

    case this.ArrivalBehaviours.AutoConfirmation:
        /* Logic to handle a an auto confirmation, whatever that may be */
        break;

    default:
        ALERT_MESSAGE(this.vName,"arrival script –
invalid behaviour requested",RequestedBehaviour.toString());

    }

    /* <ARRIVE /> - Do not remove or alter this line */

    return(true);
},

```

Okay, you have now formally defined the different arrival behaviours that your screen supports and the code required to implement them.

How are they used?

Well first, you might use them within the screen's own script instead of using payloads.

Imagine a button click that requests a page search operation like this ...

```
SENDKEY(KeyPageUp,"SEARCHNEXT");
```

Classically this sends the key stroke to the server and includes a payload so that the arrival script knows what to do with when the form arrives back again.

Note: If you have done this you probably already have an arrival script exhibiting at least two different behaviours.

Now you would code this instead:

```
this.RequestedArrivalBehaviour = this.ArrivalBehaviours.SearchNext;  
SENDKEY(KeyPageUp);
```

Note: Why do this? It seems like more work. Well you have already gained one advantage. Say you coded `SENDKEY(KeyPageUp,"SEARCHNET")` accidentally using the payload technique. It would take you a while to debug your program to find that "SEARCHNET" is wrong and it should be "SEARCHNEXT". If you coded `ArrivalBehaviours.SeachNet` your script will fail when you execute it, telling you something is wrong instantly.

The second place you would use this is in other screens.

Say another screen (named "AnotherScreen") is going to set in motion a set of events that it knows will ultimately arrive at, or pass through, your multi-behavioural screen (named "MultiScreen").

Imagine that it also needs to make sure that "MultiScreen" performs the "auto confirmation" behaviour when it arrives.

"AnotherScreen" can contain this code:

```
var oMS = SCREEN("MultiScreen"); /* Get a reference to "MultiScreen" */  
  
oMS.RequestedArrivalBehaviour =  
oMS.ArrivalBehaviours.AutoConfirmation;  
  
<< Now execute code to start events that will go to/through "MultiScreen" >>
```

In other words, "AnotherScreen" is setting a **property** in "MultiScreen" (named `RequestedArrivalBehaviour`) that says "When you arrive, I want you to perform the auto confirmation behaviour, instead of the usual default behaviour".

Note the "MultiScreen" does not have to be a destination screen. It could equally be a junction of special screen. All it needs to be is a screen whose arrival script needs to be capable of performing different behaviours.

This technique demonstrates a very formal and structured way for screens to communicate intention between themselves. You do not need to be so formal or structured, nor to use the long names suggested.

You could simply declare this in "MultiScreen":

```
Action : 0, /* Declare the action code for arriving scripts */
```

And structure the arrival script like this:

```
Switch (Action)  
Case : 0  
Case : 1  
Case : 2  
Case : 3
```

Other code would use this.Action = 2 or SCREEN("MultiScreen").Action = 3

You could even use strings like this:

```
Action : "Default", /* Declare the action code for arriving scripts */
```

And structure the arrival script like this:

```
Switch (Action)  
Case : "Default"  
Case : "Up"  
Case : "Down"  
Case : "Jump"
```

Other code would do this.Action = "Up" or SCREEN("MultiScreen").Action = "Jump".

The declaration technique you use is immaterial and long it is structured and documented. The advantage of the formal declaration (enumeration) technique is simply that it is a very formal documentation of capabilities and that code will always fail if an incorrect value is used.

Debugging

Debug and Diagnostics

Common Scripting Errors

Tracing

Using `ALERT_MESSAGE` in Your Scripts

Debug and Diagnostics

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 [alert\(\)](#) 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 5250 Form and Turn off Busy Statebutton

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

Unable to display form

Could not complete the operation due to error 80020101

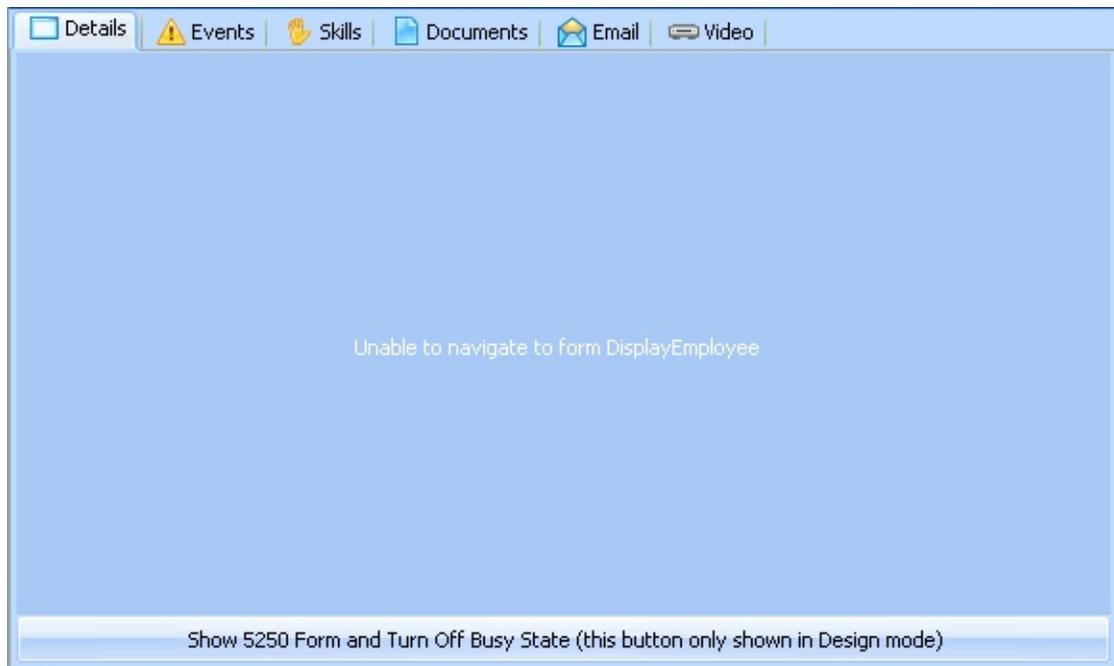
Object expected

Strange behavior in scripts

Your script does not execute at all

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 ( !(Q_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 vHandle_NAVIGATETO function. 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 5250 Form and Turn off Busy Statebutton 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.

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 and 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

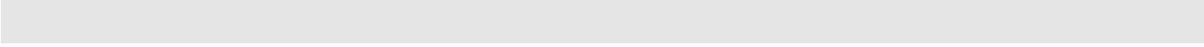
```
if (X == 2)
{
    alert("X is 2");
}
```

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.



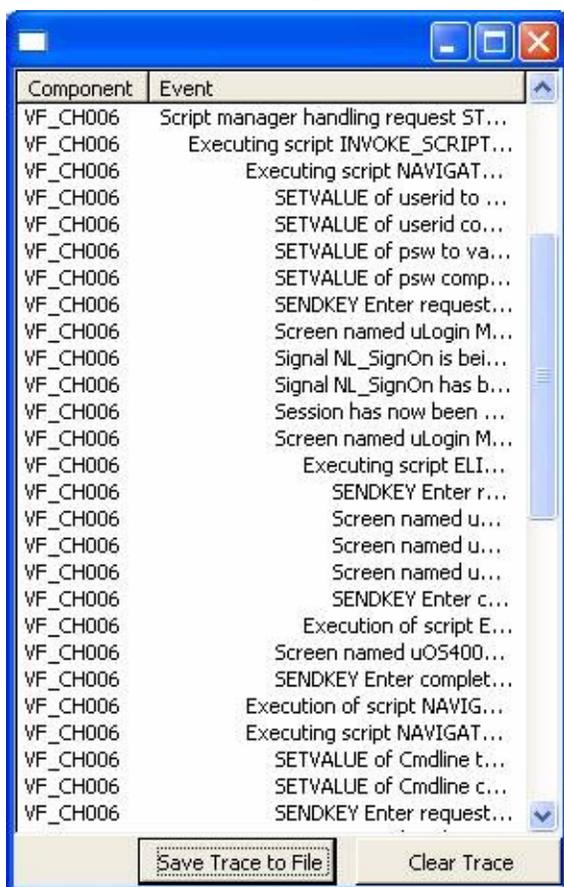
Tracing

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.

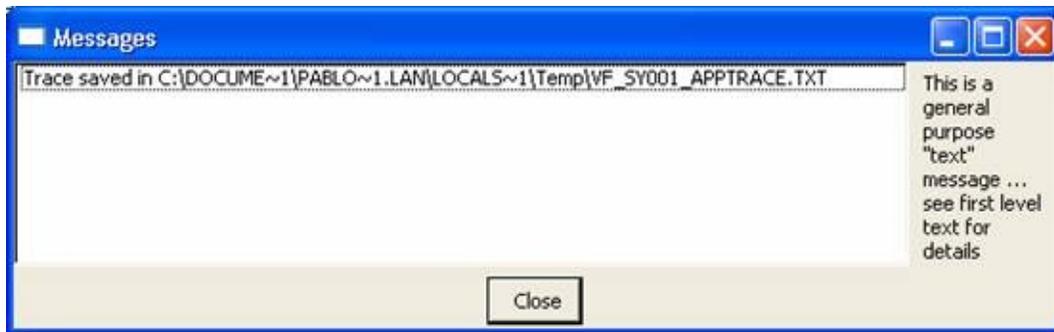
Trace statements will appear in the Trace Window.

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.



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.



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.

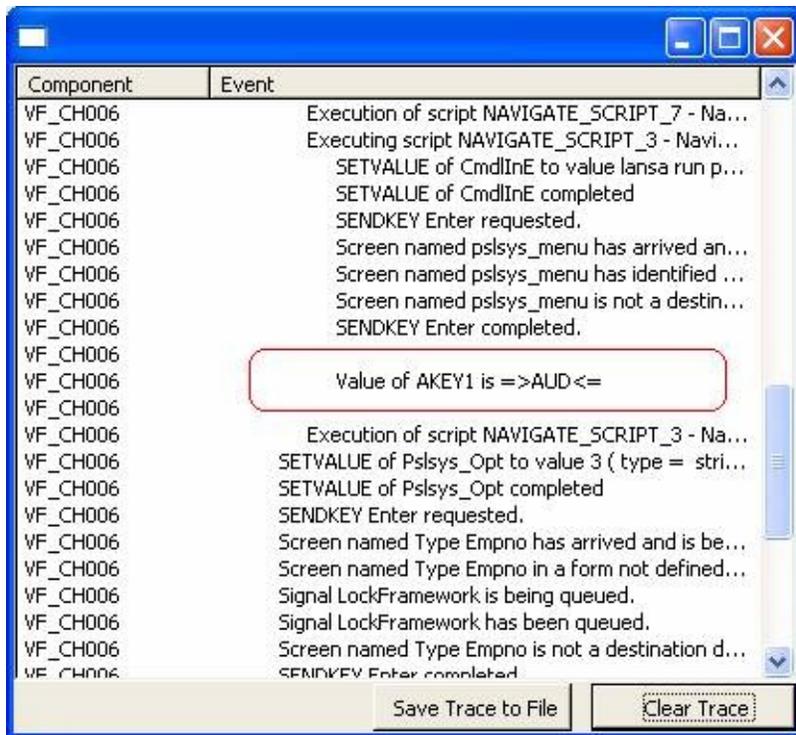
For example, this trace statement:

```
TRACE("");
```

```
TRACE("Value of AKEY1 is =>" + objListManager.AKey1[0] + "<=");
```

```
TRACE("");
```

Generates this tracing:



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:



Screen Enrichment

You may want to change the appearance of the modernized 5250 screens in your application to make them look better and more Windows-like.

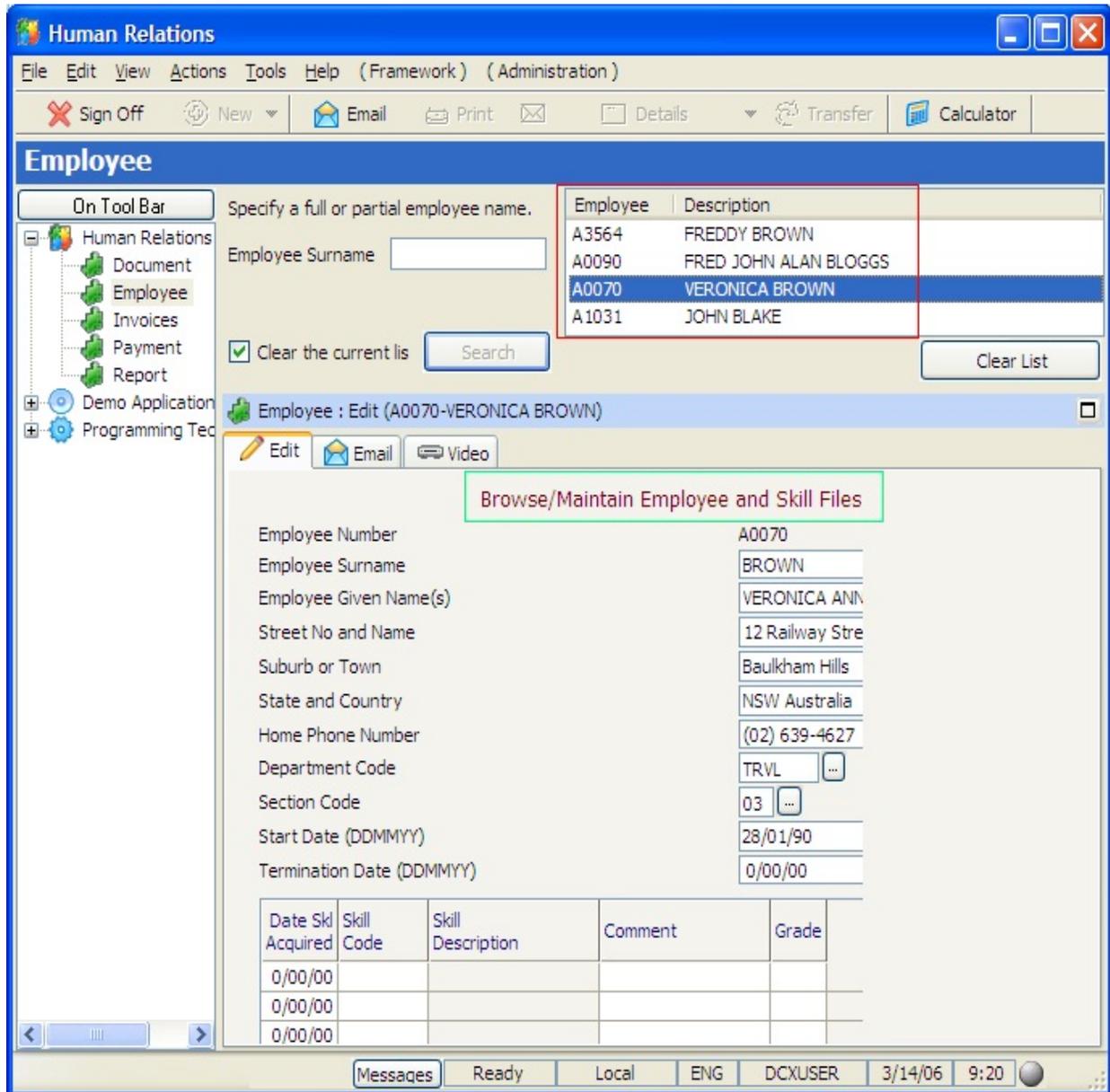
You can do simple things like hiding the screen title using RAMP layout dimensions (see [Hide screen titles in individual RAMP Screens](#)), or you can freely redesign the screens using aXes eXtensions (see [RAMP-TSAD04: Redesigning the Screen Using aXes](#)) using dropdowns, checkboxes, radio buttons, calendars, charts, hyperlinks, images and a full color palette.

There is also [The HTMLAPI Scripting Object](#) which can be used to enhance RAMP command handlers. It predates aXes eXtensions and using it requires solid JavaScript and HTML DOM skills.

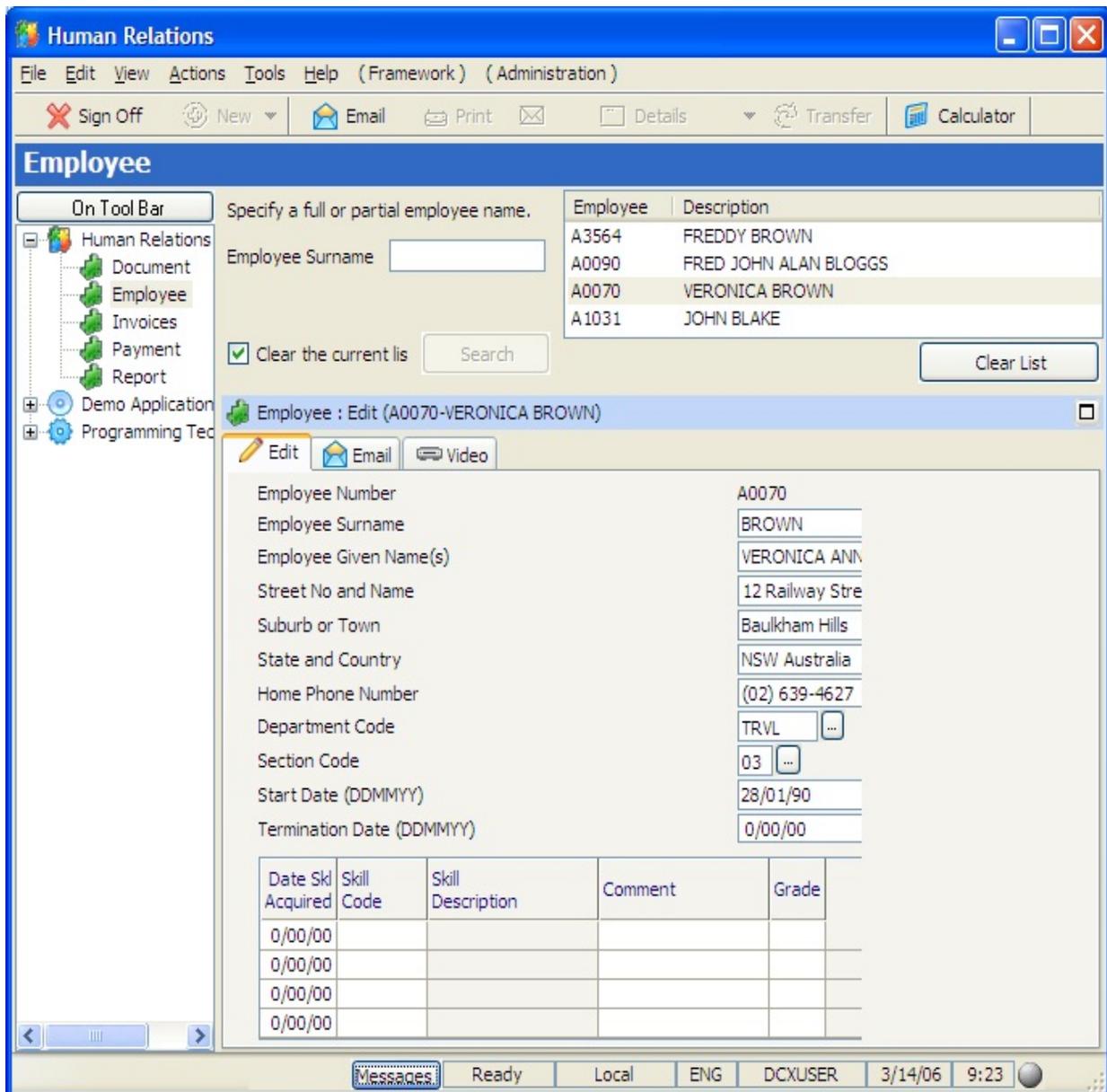
When enriching screens, please bear in mind that the space on the command handler is limited and that creating another command handler tab for additional content is easy and fast and usually a better option than trying to squeeze images, google maps or such like onto already crowded 5250 screens.

Hide screen titles in individual 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:



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-T5 5250 Session Details

Session - Default Session

Caption Default Session

User Object Name / Type 15B74640B7E7470FBA82A7F2D3F114C Verify Name

Default RAMP Layout Dimensions

Height 380 Width 700 Top 25- Left

Top Mask Height Bottom Mask Height

RAMP Screen Layout Style

Fixed Layout Flow Layout

Scroll Bars

Display Horizontal Scroll Bars Display Vertical Scroll Bars

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:

RAMP-T5 5250 Session Details

Session - Default Session

Caption Default Session

User Object Name / Type 15B74640B7E7470FBA82A7F2D3F114C Verify Name

Default RAMP Layout Dimensions

Height 380 Width 700 Top 25- Left

Top Mask Height Bottom Mask Height

RAMP Screen Layout Style

Fixed Layout Flow Layout

Scroll Bars

Display Horizontal Scroll Bars Display Vertical Scroll Bars

Masking the Title

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

title:

The image shows a configuration window titled "RAMP-TS 5250 Session" with a "Details" tab. The window is divided into several sections:

- Session - Default Session:** Contains a "Caption" field with the value "Default Session".
- User Object Name / Type:** Contains a text field with a long alphanumeric string "15B74640B7E7470FBA82A7F2D3F114C" and a "Verify Name" button.
- Default RAMP Layout Dimensions:** Contains four input fields: "Height" (380), "Width" (700), "Top" (empty), and "Left" (empty). Below these are "Top Mask Height" (25) and "Bottom Mask Height" (empty). The "Top Mask Height" field is highlighted with a red border.
- RAMP Screen Layout Style:** Contains two radio buttons: "Fixed Layout" (selected) and "Flow Layout".
- Scroll Bars:** Contains two checkboxes: "Display Horizontal Scroll Bars" and "Display Vertical Scroll Bars", both of which are unchecked.

Not applicable to RAMP Web.

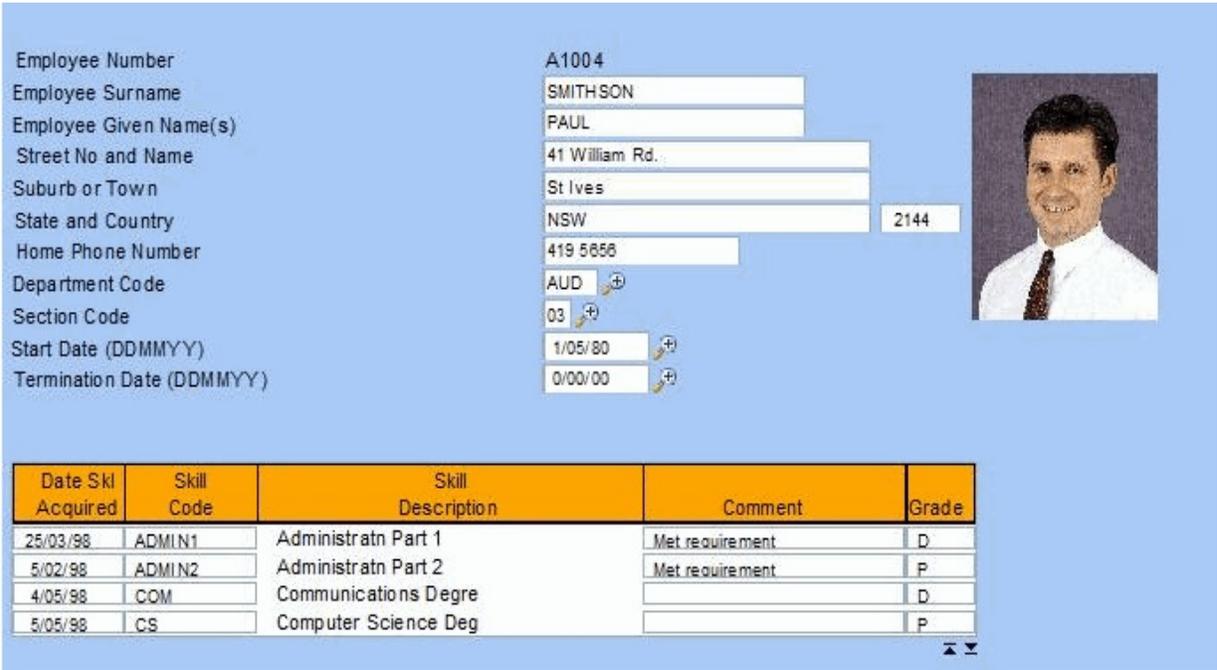
The HTMLAPI Scripting Object

The HTMLAPI scripting object can be used to dynamically modify the visual content of 5250 screens.

When RAMP-TS displays a 5250 screen, it is actually rendering a HTML document. The HTML document that represents the 5250 screen is like any other HTML document. Since it is HTML based it provides a DOM (Document Object Model) that you can access and modify from your RAMP scripts.

The HTMLAPI scripting object provides a small set of methods that allow you to access the HTML DOM of the currently displayed 5250 screen.

By using the HTML DOM you may choose to modify the appearance and behaviour of displayed 5250 screen.



The screenshot displays a 5250 screen with a light blue background. On the left, there is a list of labels for employee information. On the right, there are corresponding input fields, some of which are populated with data. A small portrait photo of a man in a white shirt and tie is visible on the right side. Below the form is a table with a yellow header and a white body, containing skill acquisition records.

Employee Number	A1004
Employee Surname	SMITHSON
Employee Given Name(s)	PAUL
Street No and Name	41 William Rd.
Suburb or Town	St Ives
State and Country	NSW 2144
Home Phone Number	419 5656
Department Code	AUD
Section Code	03
Start Date (DDMMYY)	1/05/80
Termination Date (DDMMYY)	0/00/00

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
5/02/98	ADMIN2	Administratn Part 2	Met requirement	P
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P

What do you need to know to use the HTMLAPI Scripting Object?

You need to have solid JavaScript and HTML DOM skills.

What are some of the risks involved in using the HTMLAPI Scripting Object or accessing 5250 Screen's HTML DOM?

Some of the risks you take in using the HTMLAPI Scripting Object or accessing the 5250 screen's HTML DOM include:

Future changes in the content or behaviour of the IE HTML DOM model

Future changes in the content or behaviour of the aXes 5250 HTML document layout

Upsetting or altering the behaviour of the aXes client side logic.

If you are prepared to accept, test for and manage these risks, then the HTMLAPI allows for dynamic and efficient changes to be made to the content and behaviour of 5250 RAMP-TS screens.

[HTMLAPI Usage Examples](#)

Using The HTMLAPI Scripting Object

The recommended approach to dynamically altering the visual content of 5250 screens is as follows:

Whenever possible create SHARED functions that implements generic content changes so that they can be reused from other RAMP-TS scripts. By doing this you are also somewhat minimizing the impact that any future changes to the HTML or aXes DOM models might have on you.

When content changes apply to a specific 5250 screen only, implement them in a single function inside the screens scripting object with a well defined name. By doing this the logic that alters a specific screens layout becomes somewhat standardized, easy to recognize and consolidated in one place

For example:

Example Function Name	What it might do?
SHARED.ApplyStandardLayout()	Apply generic application layout rules to any destination screen. Most destination screens would invoke this function in their arrival script, just before making the screen visible.
SHARED.AttachPromptImages()	Accepts an array of screen field names. It creates a small clickable image beside each named field. 5250 screens that display promotable fields would use this function generically.
thisScreen.ApplySpecificLayout()	Defined as a function within the definition of a specific 5250 destination screen. Applies layout changes that are specific to this screen.

HTMLAPI Usage Examples

The following examples use this 5250 destination screen which is shipped with LANSAs as process PLSYS function INQUIRE. It allows the details of an employee to be displayed and updated. It initially has an arrival script like this:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
  /* If the department input field exists on the screen, display it */

  if ( CHECK_FIELD_EXISTS("DEPARTMENT") )
  {
    SHOW_CURRENT_FORM(true);
    HIDE_5250_BUTTONS();
    SETCURSORTOFIELD("SURNAME");
    SETBUSY(false);
  }

  /* Otherwise send an F21 key to make the screen input capable */

  else
  {
    SENDKEY(KeyF21);
  }

  /* <ARRIVE /> - Do not remove or alter this line */

  return(true);
},
```

And it looks like this:

Browse/Maintain Employee and Skill Files

Employee Number : A1002
Employee Surname |SMYTHESON
Employee Given Name(s) |JOHN
Street No and Name |20 Cobbitty Avenue,
Suburb or Town |WERRINGTON.
State and Country |NSW | 2100
Home Phone Number |047 629 0442
Department Code |ADM +
Section Code |02 +
Start Date (DDMMYY) |1/01/77 +
Termination Date (DDMMYY) |0/00/00 +

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
2/05/98	ADMIN2	Administratn Part 2		F
5/05/98	CS	Computer Science Deg		P
4/05/98	ENG	English Degree		D

These fields have been named on this 5250 screen:

- SURNAME Employee Surname
- GIVENAME Employee Given Name(s)
- ADDRESS1 Street No and Name
- ADDRESS2 Suburb or Town
- ADDRESS3 State and Country
- DEPTMENT Department Code
- SECTION Section Code
- DATE_START_DDMMYY Start date
- DATE_END_DDMMYY Termination Date
- DATE_ACQ_DDMMYY Date Skill Acquired (Subfile Column)
- PageDownMarker The "+" sign indicating more data in the subfile

Note that what follows are examples only. What you do and how you work with your 5250 screens will be different. You need to adjust the approach you use to match your requirements.

[Implementing a Basic Standard Layout function](#)

[Generically Modifying a Screen via the Standard Layout function](#)

[Specifically Modifying a Screen via a Specific Layout function](#)

[Adding More Capability to the Standard Generic Handler](#)

[Modifying Subfile Headings](#)

[Modifying Fonts](#)

[Adding Images](#)

[Things to watch out for](#)

[What HTMLAPI functions are provided?](#)

Implementing a Basic Standard Layout function

First this following function is added to the uf_sy420_rts.js file as part of the SHARED scripting object:

```
var SHARED =
{
    /* ----- */
    /* Apply standard layout changes to arriving screens */
    /* ----- */

    ApplyStandardLayout : function()
    {

        /* Use the HTMLAPI to hide lines 1 and 2 on all screens */

        HTMLAPI.hideRow(1);
        HTMLAPI.hideRow(2);

    },

    Etc, Etc
}
```

We now have a standard function named SHARED,ApplyStandardLayout that can be invoked from any RAMP screen's arrival script.

For example, the arrival script of the example destination screen would be modified like this to use this new function, just before it causes the 5250 screen to be displayed:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
    /* If the department input field exists on the screen, display it */
}
```

```
if ( CHECK_FIELD_EXISTS("DEPARTMENT") )
{

    SHARED.ApplyStandardLayout();

    SHOW_CURRENT_FORM(true);
    HIDE_5250_BUTTONS();
    SETCURSORTOFIELD("SURNAME");
    SETBUSY(false);
}

/* Otherwise send an F21 key to make the screen input capable */

else
{
    SENDKEY(KeyF21);
}

/* <ARRIVE /> - Do not remove or alter this line */

return(true);
},
```

When executed now, the resulting screen looks like this:

Employee Number : A0193
 Employee Surname SMITHSON
 Employee Given Name(s) FRED
 Street No and Name 121 Cutler Ave
 Suburb or Town Windsor
 State and Country NSW 2034
 Home Phone Number (02) 546-4657
 Department Code ADM +
 Section Code 05 +
 Start Date (DDMMYY) 3/07/89 +
 Termination Date (DDMMYY) 0/00/00 +

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P
2/05/98	HIS	History Degree		F

+

Note that lines 1 and 2 on the screen are now invisible.

This is a fairly trivial generic layout rule, but you have now lain a base on which much more important rules can be generically implemented across many different 5250 screens.

Generically Modifying a Screen via the Standard Layout function

Another generic thing we might do is remove all the "+" prompt fields from the screen and strip the trailing dots from all the field labels.

To do this you could add code like this to the SHARED,ApplyStandardLayout function:

```
/* Get all elements between lines 3 and 22 */  
  
var aH = HTMLAPI.getElementsInRowRange(3,22);  
  
/* Hide all fields containing "+" signs and strip trailing dots from others */  
  
for (i = 0; i < aH.length; i++ )  
{  
    var oH = aH[i];  
  
    if ((oH != null) && (typeof(oH.tagName) != "undefined"))  
    {  
        var fIsINPUTField = ((oH.tagName == "INPUT") || (oH.tagName == "TEXTAREA"))  
  
        if ( !(fIsINPUTField) ) /* This NOT an input field on the screen */  
        {  
            if (oH.innerText == "+") HTMLAPI.hideElement(oH);  
            else HTMLAPI.stripTrailingDots(oH);  
        }  
    }  
}
```

When executed the example 5250 now looks like this:

Employee Number A1004
 Employee Surname SMITHSON
 Employee Given Name(s) PAUL
 Street No and Name 41 William Rd.
 Suburb or Town St Ives
 State and Country NSW 2144
 Home Phone Number 419 565 6
 Department Code AUD
 Section Code 03
 Start Date (DDMMYY) 1/05/80
 Termination Date (DDMMYY) 0/00/00

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
5/02/98	ADMIN2	Administratn Part 2	Met requirement	P
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P

Specifically Modifying a Screen via a Specific Layout function

A 5250 screen specific thing we can do is to add scrolling buttons to a subfile it displays.

In the script associated with this example screen, three new functions could be added like this:

```
/* Apply layout changes specific to this screen */

ApplySpecificLayout : function()
{
  if (CHECK_FIELD_EXISTS("PageDownMarker"))

HTMLAPI.insertSubFileScrollers("/ts/skins/images/pageup.gif",this.HandlePa
  else
    HTMLAPI.insertSubFileScrollers("/ts/skins/images/pageup.gif",this.Han
},

/* Handle clicks on the subfile scroller images images */

HandlePageDown: function() { EXECUTE_BUTTON_SCRIPT(KeyPageDc
HandlePageUp: function() { EXECUTE_BUTTON_SCRIPT(KeyPageUp);
```

And the arrival script part of the screen definition is modified to invoke this new logic every time a screen arrives:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
  var bReturn = true;

  /* If the department input field exists on the screen, display it */
```

```
if ( CHECK_FIELD_EXISTS("DEPARTMENT") )  
{
```

```
    SHARED.ApplyStandardLayout();  
    this.ApplySpecificLayout();
```

etc, etc

When executed the example 5250 screen now looks like this:

The screenshot shows a data entry form with the following fields and values:

Employee Number	A1004
Employee Surname	SMITHSON
Employee Given Name(s)	PAUL
Street No and Name	41 William Rd.
Suburb or Town	St Ives
State and Country	NSW 2144
Home Phone Number	419 565 6
Department Code	AUD
Section Code	03
Start Date (DDMMYY)	1/05/80
Termination Date (DDMMYY)	0/00/00

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
5/02/98	ADMIN2	Administratn Part 2	Met requirement	P
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P

At the bottom right of the table, there are two small icons: a left-pointing arrow and a right-pointing arrow, indicating page navigation.

Note the page up and down clickable images appearing at the bottom of the subfile. When clicked they invoke the handler functions HandlePageUp and HandlePageDown, which then send page up / down keystrokes to the server.

Note: They do this by executing the vHandle_BUTTONCLICK function, so you need to make sure that it can handle the page up and page down keys correctly.

Adding More Capability to the Standard Generic Handler

The RAMP-TS session used in this example has a special fields handling table like this:

Special Field Handling			
	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)
1	DEPARTMENT	F4	DF_PRM04
2	SECTION	F4	DF_PRM04
3	DATE_*	F4	DF_PRM07
4		F4	
5		F4	

If you do not understand what this means you should complete the special field handling tutorial.

The special field handling table enables automatic prompting of these fields on the example screen like this:

Employee Number	A1008
Employee Surname	SNEDDON
Employee Given Name(s)	ALLAN
Street No and Name	22 Railway Parade,
Suburb or Town	KOGARAH
State and Country	NSW. 2160
Home Phone Number	476 2198
Department Code	AUD
Section Code	01
Start Date (DDMMYY)	200309
Termination Date (DDMMYY)	0/00/

Date Skl Acquired	Skill Code	Skill Description
25/03/98	ADMIN1	Administratn Part 1
1/05/98	ADMIN2	Administratn Part 2
5/05/98	CS	Computer Science Deg
5/02/98	INTRO	Company Introduction

March 2009						
Mon	Tue	Wed	Thu	Fri	Sat	Sun
23	24	25	26	27	28	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5
Today: 20/03/2009						

Here the user has pressed F4 when the cursor was positioned in the Start Date field. The special field handler DF_PRM07 causes a calendar to appear,

allowing the user to select a date.

This only happens when the user positions into the promptable field and uses the F4 function key.

The F4 prompt logic can be generically extended further via the HTMLAPI and by using your generic SHARED.ApplyStandardLayout function.

First, modify SHARED.ApplyStandardLayout to receive an optional parameter like this:

```
ApplyStandardLayout : function(aPromptFields)
```

Then add code like this example to the ApplyStandardLayout

```
/* Insert prompting images */  
  
if (aPromptFields != null)  
{  
  for (i = 0; i < aPromptFields.length; i++)  
  {  
    oH = HTMLAPI.getElementByName(aPromptFields[i]);  
    if (oH != null)  
    {  
      oI = HTMLAPI.insertImage(oH,"/ts/skins/images/zoom_in_18x18.g  
      oI.PromptFieldName = aPromptFields[i];  
    }  
  }  
}
```

By checking `aPromptFields == null` you redesign allows for the parameter to be optional. Callers do not need to pass it.

The SHARED object also needs to have a function added to handle clicking on the images created, like this example:

```

/* ----- */
/* Handle clicking on a prompt image      */
/* ----- */

HandlePromptImageClick : function(oE)
{
  var oI = oE.srcElement;
  if (typeof(oI.PromptFieldName) != "undefined")
  {
    SETCURSORTOFIELD(oI.PromptFieldName);
    EXECUTE_BUTTON_SCRIPT(KeyF4);
  }
},

```

Finally, the example 5250 destination screen that is using SHARED.ApplyStandardLayout needs to be modified to pass an array of promptable fields.

First the array is declared like this (at the start of the scripting code):

```

aPromptFields :
Array("DEPARTMENT","SECTION","DATE_START_DDMMYY","DATE_EN

```

The call to SHARED.ApplyStandardLayout is modified

```

SHARED.ApplyStandardLayout(this.aPromptFields);

```

The resulting 5250 screen looks like this:

Employee Number A1002

Employee Surname SMYTHESON

Employee Given Name(s) JOHN

Street No and Name 20 Cobbitty Avenue,

Suburb or Town WERRINGTON.

State and Country NSW 2100

Home Phone Number 047 629 0442

Department Code ADM 

Section Code 05 

Start Date (DDMMYY) 180309 

Termination Date (DDMMYY) 100309 

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
2/05/98	ADMIN2	Administratn Part 2		F
5/05/98	CS	Computer Science Deg		P
4/05/98	ENG	English Degree		D

Note the small images now appearing beside the promptable fields.

The user can click on the image to prompt the field, or they can position into the field and press F4. The result is the same.

Clicking on the image actually executes the current screen's vHandle_BUTTONCLICK function, so it needs to be able to handle the F4 key.

Modifying Subfile Headings

One other feature that the HTMLAPI provides dynamically alters subfile headings. Typically it can be used in a generic fashion for any 5250 screen. If code like this is added to the standard SHARED layout function ApplyStandardLayout:

```
/* Adjust the subfile headings */
```

```
HTMLAPI.adjustSubFileHeadings("Arial","8pt");
```

Then the resulting 5250 screen now looks like this:

Employee Number	A1007
Employee Surname	SNELL
Employee Given Name(s)	GEORGE
Street No and Name	6 Anthony Avenue,
Suburb or Town	PADSTOW.
State and Country	NSW. 2164
Home Phone Number	764 3562
Department Code	AUD
Section Code	01
Start Date (DDMMYY)	1/03/85
Termination Date (DDMMYY)	0/00/00

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
1/05/98	ADMIN2	Administratn Part 2		P
5/05/98	CS	Computer Science Deg		P
5/02/98	INTRO	Company Introduction	Met requirement	P

By making this change in your SHARED.ApplyStandardLayout:

```
/* Adjust the subfile headings */
```

```
HTMLAPI.adjustSubFileHeadings("Arial","8pt","orange','left','1px solid l
```

You could cause 5250 subfile to look like this

Employee Number	A1004
Employee Surname	SMITHSON
Employee Given Name(s)	PAUL
Street No and Name	41 William Rd.
Suburb or Town	St Ives
State and Country	NSW 2144
Home Phone Number	419 565 6
Department Code	AUD
Section Code	03
Start Date (DDMMYY)	1/05/80
Termination Date (DDMMYY)	0/00/00

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
5/02/98	ADMIN2	Administratn Part 2	Met requirement	P
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P

Note: The 'shipped' intention with sub files is to make them look somewhat like list views or grids, which is exactly what this orange example does not do.

Modifying Fonts

Finally, here is an example of a modified SHARED layout function that changes the font of all fields on the form to use 8pt Arial. The example loop used earlier in SHARED.ApplyStandardLayout could be changed to be like this example:

```
/* Get all elements between lines 3 and 22 */

var aH = HTMLAPI.getElementsInRowRange(3,22);

/* Hide all fields containing "+" signs and strip trailing dots from others */

for (i = 0; i < aH.length; i++ )
{
    var oH = aH[i];

    if ((oH != null) && (typeof(oH.tagName) != "undefined"))
    {
        var fIsINPUTField = ((oH.tagName == "INPUT") || (oH.tagName == "TEXT"))

        if (fIsINPUTField) HTMLAPI.applyFont(oH,"Arial","7pt");
        else
        {
            if (oH.innerText == "+") HTMLAPI.hideElement(oH);
            else
            {
                HTMLAPI.stripTrailingDots(oH);
                HTMLAPI.applyFont(oH, "Arial","8pt");
            }
        }
    }
}
}
```

The resulting 5250 screen now looks like this:

Employee Number: A1004

Employee Surname: SMITHSON

Employee Given Name(s): PAUL

Street No and Name: 41 William Rd.

Suburb or Town: St Ives

State and Country: NSW 2144

Home Phone Number: 419 5656

Department Code: AUD

Section Code: 03

Start Date (DDMMYY): 1/05/80

Termination Date (DDMMYY): 0/00/00

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
5/02/98	ADMIN2	Administratn Part 2	Met requirement	P
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P

It started out as:

Browse/Maintain Employee and Skill Files

Employee Number : A1002

Employee Surname : SMYTHESON

Employee Given Name(s) : JOHN

Street No and Name : 20 Cobbitty Avenue,

Suburb or Town : WERRINGTON.

State and Country : NSW 2100

Home Phone Number : 047 629 0442

Department Code : ADM +

Section Code : 02 +

Start Date (DDMMYY) : 1/01/77 +

Termination Date (DDMMYY) : 0/00/00 +

Date Skl Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
2/05/98	ADMIN2	Administratn Part 2		F
5/05/98	CS	Computer Science Deg		P
4/05/98	ENG	English Degree		D

Adding Images

To add an image to your RAMP command handler, put this example generic code into your SHARED object:

```
var SHARED =
{

    /* ----- */
    /* Handle clicking on a prompt image          */
    /* ----- */

    oFloatingImage : null,

    InsertImage : function(sBesideField,sSource,iHeight,iWidth,iHOffset,iVOffs
    {
        var oE = HTMLAPI.getElementbyName(sBesideField);

        if (oE == null) return;

        var oC = HTMLAPI.getcontainerDIV(oE);

        if (oC == null) return;

        if (this.oFloatingImage == null)
        {
            this.oFloatingImage = oC.ownerDocument.createElement("
<IMG style='position:absolute; visibility:hidden; display:none;' >");
            oC.ownerDocument.body.insertAdjacentElement("beforeEnd",this.oFloa
        }

        this.oFloatingImage.src          = sSource;
        this.oFloatingImage.style.pixelTop   = oC.style.pixelTop + iVOffset;
        this.oFloatingImage.style.pixelLeft  = oC.style.pixelLeft + oC.style.pixelV
        this.oFloatingImage.style.pixelHeight = iHeight;
        this.oFloatingImage.style.pixelWidth = iWidth;
        this.oFloatingImage.style.visibility = "visible";
        this.oFloatingImage.style.display   = "inline";
```

```
return;  
},
```

And add a new line to your standard layout function to make sure the image disappears as new screens arrive:

```
ApplyStandardLayout : function(aPromptFields)  
{  
  
    /* Drop any floating images left around from before */  
  
    if (this.oFloatingImage != null) { this.oFloatingImage.style.visibility = "hid
```

You have now added you own completey generic InsertImage capability to any RAMP-TS 5250 screen. Try it out by adding this line to the arrival script of a screen:

```
SHARED.InsertImage("SURNAME", "/ts/skins/images/TestImage1.gif", 123, 10
```

And you get this:

The screenshot shows an employee information form with the following fields and values:

Employee Number	A1004
Employee Surname	SMITHSON
Employee Given Name(s)	PAUL
Street No and Name	41 William Rd.
Suburb or Town	St Ives
State and Country	NSW 2144
Home Phone Number	419 5658
Department Code	AUD
Section Code	03
Start Date (DDMMYY)	1/05/80
Termination Date (DDMMYY)	0/00/00

Below the form is a table of skills acquired:

Date Skil Acquired	Skill Code	Skill Description	Comment	Grade
25/03/98	ADMIN1	Administratn Part 1	Met requirement	D
5/02/98	ADMIN2	Administratn Part 2	Met requirement	P
4/05/98	COM	Communications Degre		D
5/05/98	CS	Computer Science Deg		P

A portrait photo of Paul Smithson is displayed to the right of the form.

The `SHARED.InsertImage()` can now be reused anywhere in your application. Do not forget to set authority correctly on an new files in Axes folder.

Remember that adding a new command handler tab for the image would be a quicker and easier solution.

Things to watch out for

Here are some things to watch out for:

Understand and accept the risks that low level access to a DOM structure inherently involves (see the preceding section for more about this).

Check that you are not impacting the performance of your application. Doing such a check is easy. Comment out or disable your logic. Do some timing tests using a PC that has a performance profile like a typical end user's PC. Then repeat the tests with your logic enabled.

Do not become over-focused on a 5250 screen. For example, imagine you have a customer inquiry 5250 screen. You want to add a Google Maps image to your application to show the customer's location. There are two ways to do this:

Add an IFRAME to your 5250 screen with the required JavaScript logic.

Simply add another tab called "Map Location" to your VLF application.

Of these, the latter would be simpler and easier to implement. It is also 'on demand' rather than cluttering up your screen with information that most people do not need to see most of the time.

What HTMLAPI functions are provided?

HTMLAPI.getElementbyName

Parameters:

Description	Type	Optional/Mandatory	Default Value
Element Name	String	Mandatory	
Index	Integer	Optional	0

Returns: A reference to the HTML element or null if not found.

HTMLAPI.getElementbyLocation

Parameters:

Description	Type	Optional/Mandatory	Default Value
Row	Integer	Mandatory	
Column	Integer	Mandatory	

Returns: A reference to the HTML element or null if not found.

HTMLAPI.getElementsinRowRange

Parameters:

Description	Type	Optional/Mandatory	Default Value
Low Row	Integer	Mandatory	
High Row	Integer	Optional	Low Row

Returns: An array of references to the HTML elements.

HTMLAPI.getElementsInColumnRange

Parameters:

Description	Type	Optional/Mandatory	Default Value
Low Column	Integer	Mandatory	
High Column	Integer	Optional	Low Column

Returns: An array of references to the HTML elements.

HTMLAPI.showElement

Parameters:

Description	Type	Optional/Mandatory	Default Value
HTML element	HTML element reference	Mandatory	

Returns: null.

HTMLAPI.hideElement

Parameters:

Description	Type	Optional/Mandatory	Default Value
HTML element	HTML element reference	Mandatory	

Returns: null.

HTMLAPI.hideRow

Parameters:

Description	Type	Optional/Mandatory	Default Value
Row number	Integer	Mandatory	

Returns: null.

HTMLAPI.insertImage

Parameters:

Description	Type	Optional/Mandatory	Default Value
HTML element that image is to be inserted after	HTML element reference	Mandatory	
Image source	String	Mandatory	
Click event handler	Function	Optional	null
Image height	Integer	Optional	Image's own height
Image width	Integer	Optional	Image's own width
Horizontal offset	Integer	Optional	2
Vertical offset	Integer	Optional	0

Returns: Reference to the HTML element created or null in error situations

elements.

HTMLAPI.applyFont

Parameters:

Description	Type	Optional/Mandatory	Default Value
HTML element to which font is to be applied	HTML element reference	Mandatory	
Font Family	String	Mandatory	
Font Size	String	Mandatory	

Returns: null.

HTMLAPI.getcontainerDIV

Parameters:

Description	Type	Optional/Mandatory	Default Value
HTML element whose absolutely positioned container DIV is to found	HTML element reference	Mandatory	

Returns: Reference to the absolutely positioned container DIV or null if not found.

HTMLAPI.stripTrailingDots

Parameters:

Description	Type	Optional/Mandatory	Default Value
--------------------	-------------	---------------------------	----------------------

HTML element from which dots are to be stripped HTML element reference Mandatory

Returns: null.

HTMLAPI.insertSubFileScrollers

Parameters:

Description	Type	Optional/Mandatory	Default Value
Page up image to use	String	Optional	Null
Page up click handler	Function	Optional	Null
Page Down image to use	String	Optional	Null
Page Down click handler	Function	Optional	Null
Images Height	Integer	Optional	12
Images Width	Integer	Optional	12
Horizontal offset	Integer	Optional	0
Vertical offset	Integer	Optional	0

Returns: null.

HTMLAPI.adjustSubFileHeadings

Parameters:

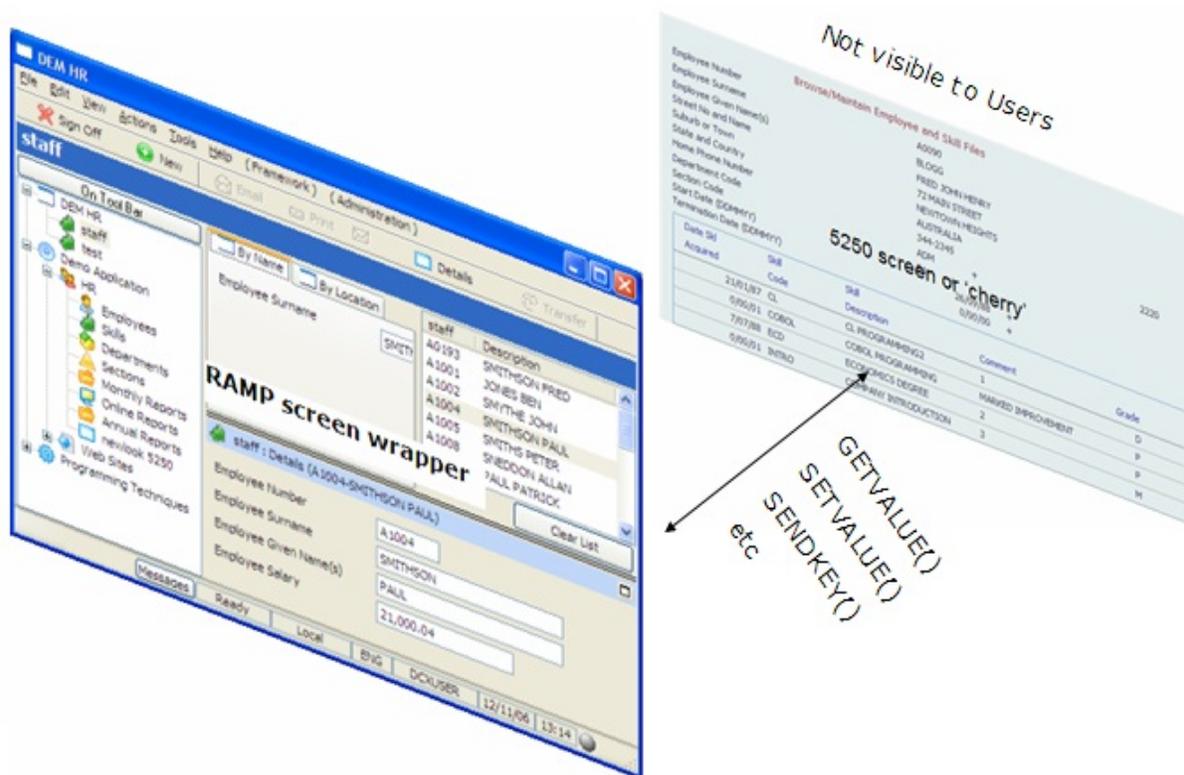
Description	Type	Optional/Mandatory	Default Value
Font to use	String	Optional	Lucida Console
Font Size	String	Optional	8pt
Background Color	String	Optional	Buttonface
Text Alignment	String	Optional	center
Border Style	String	Optional	1px solid darkgray

Returns: null.

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

EventsMethodsExamples

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) Displayposition(3)
```

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_Ancestors.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

Usually you will invoke [MakerampTSavailable Method](#) inside the uExecute

method of your command for the first time:

```
#myscreen_wrapper.MakerampTSAvailable
```

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 RampTSAvailable event routine.

Listen to the RampTSAvailable event

Once RAMP has connected and it's ready to be interacted with it will signal back to the command in this event. It means you are ready to start navigation. For example:

```
Invoke Method(#myscreen_wrapper.navigatetoscreen)  
Name(EMPLOYEE_SKILLS)
```

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

RampTSAvailable Event

vHandleArrive 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:

Type	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_UNKNOWN_FORM	During navigation, an undefined form was detected.	
-----------------	--	--

Remarks

It is 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 5250 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

```

RampTSAvailable Event

RAMP has signaled it is interactive.

Parameters

None

Remarks

Start your navigation here.

Example

```
EvtRoutine Handling(#myscreen_wrapper.RampTSAvailable)
  Invoke Method(#myscreen_wrapper.navigatetoscreen)
  Name(EMPLOYEE_SKILLS)
Endroutine
```

Also see [MakerampTSAvailable Method](#).

vHandleArrive Event

A screen has arrived.

ArrivedScreen	Char 256	String that specifies the name of the arrived screen.
PreviousScreen	Char 256	String that contains the name of the previous screen.
ArrivedPayload	Char 256	String that contains a payload.
ForAction	Char 256	String that was passed as a parameter to the NavigateToScreen method.
VariantName	Char 256	String that contains the variant name of the arrived screen

Remarks

This event will signal for every navigated screen.

Example

```
Evtroutine Handling(#myscreen_wrapper.vHandleArrive)
Arrivedscreen(#ArrivedScreen) Previouscreen(#PreviousScreen)
Arrivedpayload(#Payload)

Case (#ArrivedScreen)

When Value_Is(= EMPLOYEE_DETAILS)

Set Com(#SAVE_BUTTON) Enabled(false)

If (#Payload *NE UPDATE_EMPLOYEE)
```

```
#myscreen_wrapper.getvalue From('empno') Value(#empno.value)  
#myscreen_wrapper.getvalue From('surname') Value(#surname.value)
```

```
Endif
```

```
Endcase
```

```
Endroutine
```

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):

[MakerampTSavailable Method](#)

[NavigateToScreen Method](#)

[SetValue Method](#)

[GetValue Method](#)

[SendKey Method](#)

[Current_Form Method](#)

[SetCursor Method](#)

[SetCursorToField Method](#)

[Get_Form_Message Method](#)

[Check_Field_Exists Method](#)

MakerampTSavailable Method

Make RAMP interactive.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.MakerampTSavailable
```

Parameters

uObjectType	Char 32 - Optional	String that contains the user object type of the RAMP session. Required when dealing with more than one session.
uSession_Id	Char 40 - Optional	The session assigned to a destination. Defaults to *AUTO.

Return Value

None

Remarks

Invoke this method in the uExecute command handler method, there is no performance penalty in doing this. It will ensure that you can interact with RAMPTS.

Examples

```
Invoke Method(#myscreen_wrapper.MakerampTSavailable)  
uSession_Id(SESSION_A)  
Invoke Method(#myscreen_wrapper.MakerampTSavailable)  
uObjectType(HumanResources)
```

Related Topic [MakerampTSavailable Method](#).

NavigateToScreen Method

Navigate to a RAMP-TS screen.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.NavigateToScreen Name(EMPLOYEE_SKILLS)
```

Parameters

Name	Char 256 – Required	String that contains the name of the screen to navigate to.
------	------------------------	--

Return Value

None

Remarks

Once NavigateToScreen is executed your screen wrapper will receive screen arrival event signals to be handled in the vHandleArrive event routine.

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.Unavigatetoscreen) Name(EMPLOYEE_SKILLS)
```

SetValue Method

Set the value of an input field on a 5250 screen. Pass an index to set the value of an input field in a subfile.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
Setting by Name - #myscreen_wrapper.setvalue Infield(sFieldName)  
Value(vValue)
```

Parameters

Setting by Name:

InField	Char 256 – Required	String that contains the name of an input field.
Value	Variant – Required	String or number that contains the value.
Index	Integer - Optional	An Integer that specifies the subfile row of the field. Note: the specified row index must exist in the current subfile page.

Return Value

None

Remarks

Only Input fields that have been named can have their values set.

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.setvalue Infield(GIVENAME) Value(#Givename)
```

* Set the value of a field in the 3rd row of the current subfile page
#myscreen_wrapper.setvalue Infield(SFL_OPTION) Value(#SelOption)
Index(3)

GetValue Method

Get the value from a field on a screen or in subfile.

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 name of the field to get the value from.
DefaultValue	Char 256 – Optional	String that contains the default value to return when the field is not found.
Index	Integer - Optional	An Integer that specifies the subfile row of the field. Note: the specified row index must exist in the current subfile page.

Return Value

Value	Variant – Required	Returns the field value as a string or number.
-------	--------------------	--

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.getvalue From(SURNAME) Value(#surname.value)
```

```
#myscreen_wrapper.getvalue From(SURNAME) Value(#surname.value)  
Index(5) DefaultValue(*Blanks)
```

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	The property of #myscreen_wrapper that resolves to the desired key.
	–	
	Required	For a list of these properties See the SENDKEY Names in Function Key Names for SENDKEY Function in lansa049.chm.
Payload	Char	A string that contains a payload. The payload is returned in vHandleArrive
	256 -	
	Optional	

Return Value

None

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.Sendkey Key(#myscreen_wrapper.KeyPageDown)  
Payload(NEXT_PAGE)
```

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 – Required	String that contains the name of the current screen wrapper screen
------	------------------------	---

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.current_form Name(#std_txtl)
```

SetCursor Method

Positions the cursor in a given row and column of the screen.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.SetCursor RowNum(iRowNum) ColNum(iColNum)
```

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.

Return Value

None

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.setcursor Rownum(10)
```

SetCursorToField Method

Positions the cursor in a given field on a screen or subfile.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.SetCursorToField Name(SURNAME)
```

Parameters

Name	Char 256 – Required	String that specifies the name of the field to position the cursor at.
Index	Integer - Optional	An Integer that specifies the subfile row of the field. Note: the specified row index must exist in the current subfile page.

Return Value

None

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#myscreen_wrapper.setcursortofield Name(SURNAME) Index(3)
```

Get_Form_Message Method

Retrieves and routes a message in a specified screen row number.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#bMoreMsgs := #myscreen_wrapper.GetFormMessage(iRow)
```

Parameters

RowNumber	Integer – Required	Integer that specifies the screen's row number where to get the message from.
-----------	--------------------	---

Return Value

MoreMessages	Boolean	For messages presented in subfiles: true – there are more messages false – no more messages
--------------	---------	---

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#MoreMsgs := #myscreen_wrapper.Get_Form_Message(22)
```

Check_Field_Exists Method

Checks if a field is present in the current screen or subfile.

Syntax

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#bFound := #myscreen_wrapper.Check_Field_Exists(sName iInd)
```

Parameters

Name	Char 256 – Required	Integer that specifies the row number where to position the cursor.
Index	Integer – Optional	An Integer that specifies the subfile row of the field. Note: the specified row index must exist in the current subfile page.

Return Value

Found	Boolean	true – the field was found in the current screen false – the field was not found
-------	---------	---

Examples

```
Define_Com Class(#vf_sy122) Name(#myscreen_wrapper)  
Parent(#COM_OWNER)  
#FldFound := #myscreen_wrapper.Check_Field_Exists("SURNAME")  
#FldFound := #myscreen_wrapper.Check_Field_Exists("SURNAME"  
#listcount)
```

```
Dowhile (#myscreen_wrapper.check_field_exists( "SKILLCODE" #listcount
))
#myscreen_wrapper.getvalue From("skillcode") Value(#skilcode)
Index(#listcount)
#listcount += 1
Endwhile
```

Examples

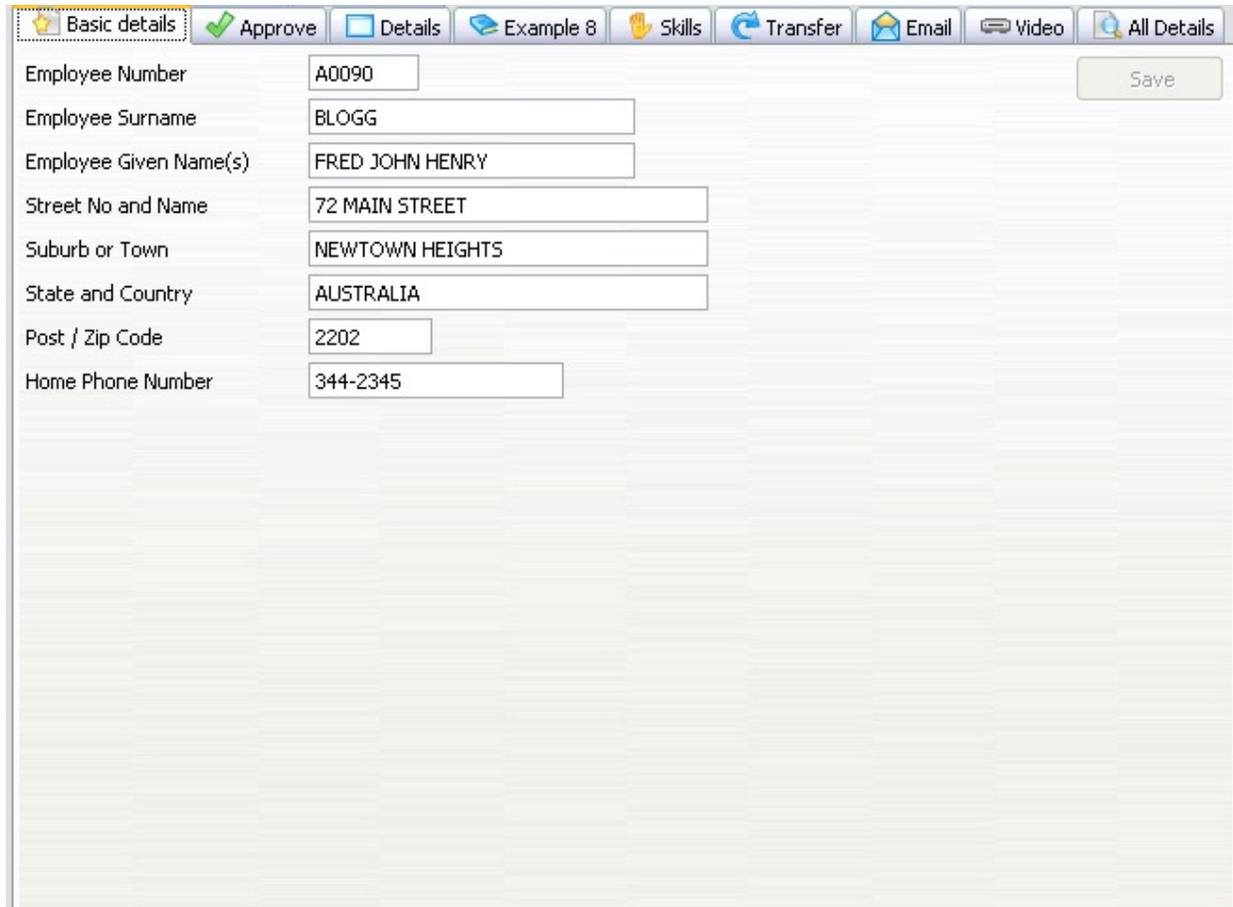
Example 1: Show Employee Details.

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.



Field	Value
Employee Number	A0090
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)

*

=====

* Simple Field and Group Definitions

*

=====

Group_By Name(#XG_HEAD) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #PHONEHME
#DEPARTMENT #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) Readonly(True)
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) 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)
```

*

=====

* To better understand this example you should be familiar with the shipped Personnel System demo.

*

* We expect the following screens to appear as part of this navigation:

*

* Login -> type in user and password -> press Enter -> (a special screen? <F3>)

* i5 Main Menu -> type lansa run process(pslsys) partition(dem) -> press Enter

* Personnel System -> type option 3 -> press Enter

* Inquire -> type the current instance employee number -> press Enter

* Browse Employee Details and Skills in OUTPUT mode -> press F21

* Browse Employee Details and Skills in INPUT mode

*

=====

*

* -----

* 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

*

* Always invoke makerampTSavailable to ensure RAMP-TS is up and running before starting a navigation

*

* You may also disable the entire form to prevent any input while RAMP is

```

navigating
* -----
-----

Mthroutine Name(uExecute) Options(*REDEFINE)

Invoke Method(#Com_Ancesor.uExecute)

#myscreen_wrapper.makerampTSAvailable

Set Com(#Save_Button) Enabled(False)

#com_owner.enabled := false
Endroutine

*
=====
* Event Handlers
*
=====

* RAMP has signalled it's ready. Invoke your navigation here.
*
* Once the navigaton starts, processing resumes in the vHandleArrive event
handler.
*
Evtroutine Handling(#myscreen_wrapper.RampTSAvailable)

Invoke Method(#myscreen_wrapper.navigatetoscreen) Name('updempskills')

Endroutine

*
* The Payload is a 256 character string sent together with a SENDKEY. Use
the Payload in the same way you would use parameters in an event.
* Assuming when 'updempskills' arrives we change some details and press
Enter or click on the SAVE button, we expect one of these 2 screens to appear:
* If the SAVE was successfull, the INQUIRE screen appears.
* If the SAVE was NOT successfull, the update employee screen will reappear

```

* Setting the Payload we determine what caused the screen 'updempskills' to arrive.

*

* 1. In the Button script of updempskills for the Enter key, we attach a payload = "UPDATE_EMPLOYEE":

*

* case KeyEnter:

* SENDKEY(KeyEnter, "UPDATE_EMPLOYEE");

* break;

* 2. In the ARRIVE script of INQUIRE, we test payload. If the Payload is UPDATE_EMPLOYEE this is telling us we have just done an Update and we are most likely to want to go back to the same screen.

*

* if (TOSTRING(oPayload) == "UPDATE_EMPLOYEE")

* {

* NAVIGATE_TO_DESTINATION("updempskills");

* }

*

Evtroutine Handling(#myscreen_wrapper.vHandleArrive)

Arrivedscreen(#CurrentScreen) Previouscreen(#PreviousScreen)

Arrivedpayload(#Payload)

Case (#CurrentScreen)

When Value_Is(= 'updempskills')

Set Com(#SAVE_BUTTON) Enabled(false)

* Payloads are destroyed when the ARRIVE script finishes executing.

Therefore, a payload of UPDATE_EMPLOYEE would most likely mean there was a validation error. Otherwise - if INQUIRE had arrived - the Payload would have been destroyed.

If (#Payload = UPDATE_EMPLOYEE)

Use Builtin(message_box_show) With_Args(ok ok info "Validation Error"
"Please correct any errors")

Else

```
* Unlock the framework
Set Com(#avFrameworkManager) Ulocked(false)

#myscreen_wrapper.getvalue From('empno') Value(#empno.value)
#myscreen_wrapper.getvalue From('surname') Value(#surname.value)
#myscreen_wrapper.getvalue From('givenname') Value(#givenname.value)
#myscreen_wrapper.getvalue From('address1') Value(#address1.value)
#myscreen_wrapper.getvalue From('address2') Value(#address2.value)
#myscreen_wrapper.getvalue From('address3') Value(#address3.value)
#myscreen_wrapper.getvalue From('homephone') Value(#phonehme.value)
#myscreen_wrapper.getvalue From('postcode') Value(#POSTCODE.value)

#com_owner.enabled := true

Endif

Endcase

Endroutine

* -----
* Handle Save
* -----

Mthroutine Name(Save)

* Set the 5250 field values to the values from this panel
#myscreen_wrapper.setvalue Infield('surname') Value(#surname.value)
#myscreen_wrapper.setvalue Infield('givenname') Value(#givenname.value)
#myscreen_wrapper.setvalue Infield('address1') Value(#address1.value)
#myscreen_wrapper.setvalue Infield('address2') Value(#address2.value)
#myscreen_wrapper.setvalue Infield('address3') Value(#address3.value)
#myscreen_wrapper.setvalue Infield('homephone') Value(#phonehme.value)
#myscreen_wrapper.setvalue Infield('postcode') Value(#POSTCODE.value)

* Send the Enter key with the payload
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyEnter)
Payload(UPDATE_EMPLOYEE)
```

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 5250 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
```

```
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) Unlocked(USER)
```

```
Unlockedmessage(#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)

#com_owner.Save

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 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.

The screenshot shows a Visual LANSA application window with the following elements:

- Navigation Bar:** Basic details, Approve, Details, Example 8, Skills (highlighted), Transfer, Email, Video, All Details.
- Employee Details Form:**
 - Employee Number: A0090
 - 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
- Skills List Table:**

Acquired	Skill Code	Skill	Comment	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
#DEPARTMENT #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(#PRIM_ATLI) Name(#ATLI_4) Attachment(Center)
Parent(#MAIN_LAYOUT)
Define_Com Class(#PRIM_ATLI) Name(#ATLI_6) Attachment(Center)
Parent(#MAIN_LAYOUT)

Define_Com Class(#PRIM_FWLI) Name(#FWLI_1)
Parent(#BODY_HEAD_FLOW)

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_LVCL) Name(#LVCL_1) Caption('Acquired')
Captiontype(Caption) Displayposition(1) Parent(#skills)
Source(#VF_ELXTS) Width(18) Widthtype(Fixed)
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(#vf_sy122) Name(#myscreen_wrapper)
Displayposition(3) Height(569) Parent(#COM_OWNER) Width(688)
```

```
*
```

```
=====
* To better understand this example you should be familiar with the shipped
Personnel System demo.
```

```
*
```

```
* We expect the following screens to appear as part of this navigation:
```

```
*
```

```
* Login -> type in user and password -> press Enter -> (a special screen?
<F3>)
```

```
* i5 Main Menu -> type lansa run process(pslsys) partition(dem) -> press
Enter
```

```
* Personnel System -> type option 3 -> press Enter
```

```
* Inquire -> type the current instance employee number -> press Enter
```

```
* Browse Employee Details and Skills in OUTPUT mode -> press F21
```

```
* Browse Employee Details and Skills in INPUT mode
```

```
*
```

```
=====
*
```

```
* 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

*

* Always invoke makerampTSavailable to ensure RAMP-TS is up and running before starting a navigation

*

* You may also disable the entire form to prevent any input while RAMP is navigating

* -----

Mthroutine Name(uExecute) Options(*REDEFINE)

Invoke Method(#Com_Ancesor.uExecute)

Invoke Method(#myscreen_wrapper.MakeRampTSAvailable)

Set Com(#Save_Button) Enabled(False)

#com_owner.enabled := false

Endroutine

*

=====

* Event Handlers

*

=====

*

* RAMP has signalled it's ready. Invoke your navigation here. Also, clear the

list and panel fields.

*

* Once the navigation starts, processing resumes in the vHandleArrive event handler.

*

Eventroutine Handling(#myscreen_wrapper.RampTSAvailable)

Clr_List Named(#skills)

#XG_HEAD := *null

Invoke Method(#myscreen_wrapper.navigatetoscreen) Name('updempskills')

Endroutine

*

* The Payload is a 256 character string sent together with a SENDKEY. Use the Payload in the same way you would use parameters in an event.

* Assuming when 'updempskills' arrives we change some details and press Enter or click on the SAVE button, we expect one of these 2 screens to appear:

* If the SAVE was successful, the INQUIRE screen appears.

* If the SAVE was NOT successful, the update employee screen will reappear

* Setting the Payload we determine what caused the screen 'updempskills' to arrive.

*

* 1. In the Button script of updempskills for the Enter key, we attach a payload = "UPDATE_EMPLOYEE":

*

* case KeyEnter:

* SENDKEY(KeyEnter, "UPDATE_EMPLOYEE")

* break

* 2. In the ARRIVE script of INQUIRE, we test payload. If the Payload is UPDATE_EMPLOYEE this is telling us we have just done an Update and we are most likely to want to go back to the same screen.

*

* if (TOSTRING(oPayload) == "UPDATE_EMPLOYEE")

* {

* NAVIGATE_TO_DESTINATION("updempskills")

* }

*

```
EvtRoutine Handling(#myscreen_wrapper.vHandleArrive)
Arrivedscreen(#CurrentScreen) PreviousScreen(#PreviousScreen)
Arrivedpayload(#Payload)
```

```
Define_Com Class(#prim_boln) Name(#MoreRecords)
```

```
Case (#CurrentScreen)
```

```
When Value_Is(= 'updempskills')
```

```
Set Com(#SAVE_BUTTON) Enabled(false)
```

* Payloads are destroyed when the ARRIVE script finishes executing.
Therefore, a payload of UPDATE_EMPLOYEE would most likely mean there was a validation error. Otherwise - if INQUIRE had arrived - the Payload would have been destroyed.

```
If (#Payload = UPDATE_EMPLOYEE)
```

```
Use Builtin(message_box_show) With_Args(ok ok info "Validation Error"
"Please correct any errors")
```

```
Else
```

* Unlock the framework - no harm done if it wasn't locked

```
Set Com(#avFrameworkManager) Unlocked(false)
```

* Get the current instance and the values of the panel fields

```
Invoke Method(#avListManager.GetCurrentInstance) Akey1(#deptment)
Akey2(#section)
```

* If there are no entries in the list or have sent a PageDown key, get the subfile page currently on the hidden 5250

```
If ((#skills.entries *LE 0) Or (#Payload = NEXT_PAGE))
```

```
#MoreRecords := #com_owner.uGetSubfilePage
```

* There are more records in the subfile, send a pagedown. Processing will continue in this same event handler once the same screen with the next subfile

```

page has arrived
If (#MoreRecords)

#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyPageDown)
Payload(NEXT_PAGE)

* When all the records have been added to the list view we can re enable the
panel
Else
#com_owner.enabled := true
#myscreen_wrapper.getvalue From('empno') Value(#empno.value)
#myscreen_wrapper.getvalue From('surname') Value(#surname.value)
#myscreen_wrapper.getvalue From('givenname') Value(#givenname.value)
#myscreen_wrapper.getvalue From('address1') Value(#address1.value)
#myscreen_wrapper.getvalue From('address2') Value(#address2.value)
#myscreen_wrapper.getvalue From('address3') Value(#address3.value)
#myscreen_wrapper.getvalue From('homephone') Value(#phonehme.value)
#myscreen_wrapper.getvalue From('postcode') Value(#POSTCODE.value)
Endif

Else
* We can assume that there has been a successfull update so update the
instance list.
Invoke Method(#avListManager.UpdateListEntryData) Akey1(#Deptment)
Akey2(#Section) Akey3(#Empno.value) Visualid2(#surname + " " +
#givenname) Businessobjectype(EMPLOYEE)

Endif

Endif

Endcase
Set Com(#myscreen_wrapper) Visible(False)
Endroutine

* Traverse the skills subfile/browselist
* The technique used here consists of picking one field we know it's in the
subfile, in this case "dateacq" and while there it is present on the screen, get
all teh fields.

```

* Fields in RAMP-TS subfiles are indexed starting from 1. A subfile page with 7 rows will have 7 instances of each of the fields in the subfile. Here we increase #listcount and use it to get the value of a field.

* Note that you can specify a default value when using the getvalue method.
Mthroutine Name(uGetSubfilePage)

```
Define_Map For(*result) Class(#prim_boln) Name(#NextPage)
Define Field(#MoreVal) Type(*char) Length(1)
```

```
#listcount := 1
```

```
#NextPage := false
```

```
Dowhile (#myscreen_wrapper.check_field_exists( "dateacq" #listcount ))
```

```
#myscreen_wrapper.getvalue From("dateacq") Value(#vf_eltxts)
```

```
Defaultvalue(#ddmmyy) Index(#listcount)
```

```
#myscreen_wrapper.getvalue From("skillcode") Value(#skilcode)
```

```
Index(#listcount)
```

```
#myscreen_wrapper.getvalue From("skildesc") Value(#skildesc)
```

```
Index(#listcount) Defaultvalue("Default value")
```

```
#myscreen_wrapper.getvalue From("comment") Value(#comment)
```

```
Index(#listcount)
```

```
#myscreen_wrapper.getvalue From("grade") Value(#grade) Index(#listcount)
```

* You can put some tracing

```
#com_owner.avframeworkmanager.avRecordTrace Component(#com_owner)
```

```
Event("Adding entry = " + #vf_eltxts + ", " + #skilcode + ", " + #skildesc + ", " + #grade)
```

```
Add_Entry To_List(#skills)
```

```
#listcount += 1
```

```
Endwhile
```

* when identifying this screen we set the name of the "+" sign = "moreindicator". The presence of that field in the last row of the subfile tells us whether there is another page. The last row is one less than the current value of #listcount.

```

#listcount -= 1

If (#myscreen_wrapper.check_field_exists( "moreindicator" #listcount ))
#myscreen_wrapper.getvalue From("moreindicator") Value(#MoreVal)
Index(#listcount)
#NextPage := (#MoreVal.trim *NE "")
Endif

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 5250 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

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
* -----

Evtoutine 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)

```
Unlockedmessage(#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)
```

```
* Set the 5250 field values to the values from this panel  
#myscreen_wrapper.setvalue Infield('surname') Value(#surname.value)  
#myscreen_wrapper.setvalue Infield('givenname') Value(#givenname.value)  
#myscreen_wrapper.setvalue Infield('address1') Value(#address1.value)  
#myscreen_wrapper.setvalue Infield('address2') Value(#address2.value)  
#myscreen_wrapper.setvalue Infield('address3') Value(#address3.value)  
#myscreen_wrapper.setvalue Infield('homephone') Value(#phonehme.value)  
#myscreen_wrapper.setvalue Infield('postcode') Value(#POSTCODE.value)
```

```
* Send the Enter key with the payload  
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyEnter)  
Payload(UPDATE_EMPLOYEE)
```

```
Endroutine
```

```
* -----  
* Handle Termination  
* -----
```

```
Mthroutine Name(uTerminate) Options(*REDEFINE)
```

```
* Clean up the collection of fields on the panel  
Invoke Method(#PanelFields.RemoveAll)
```

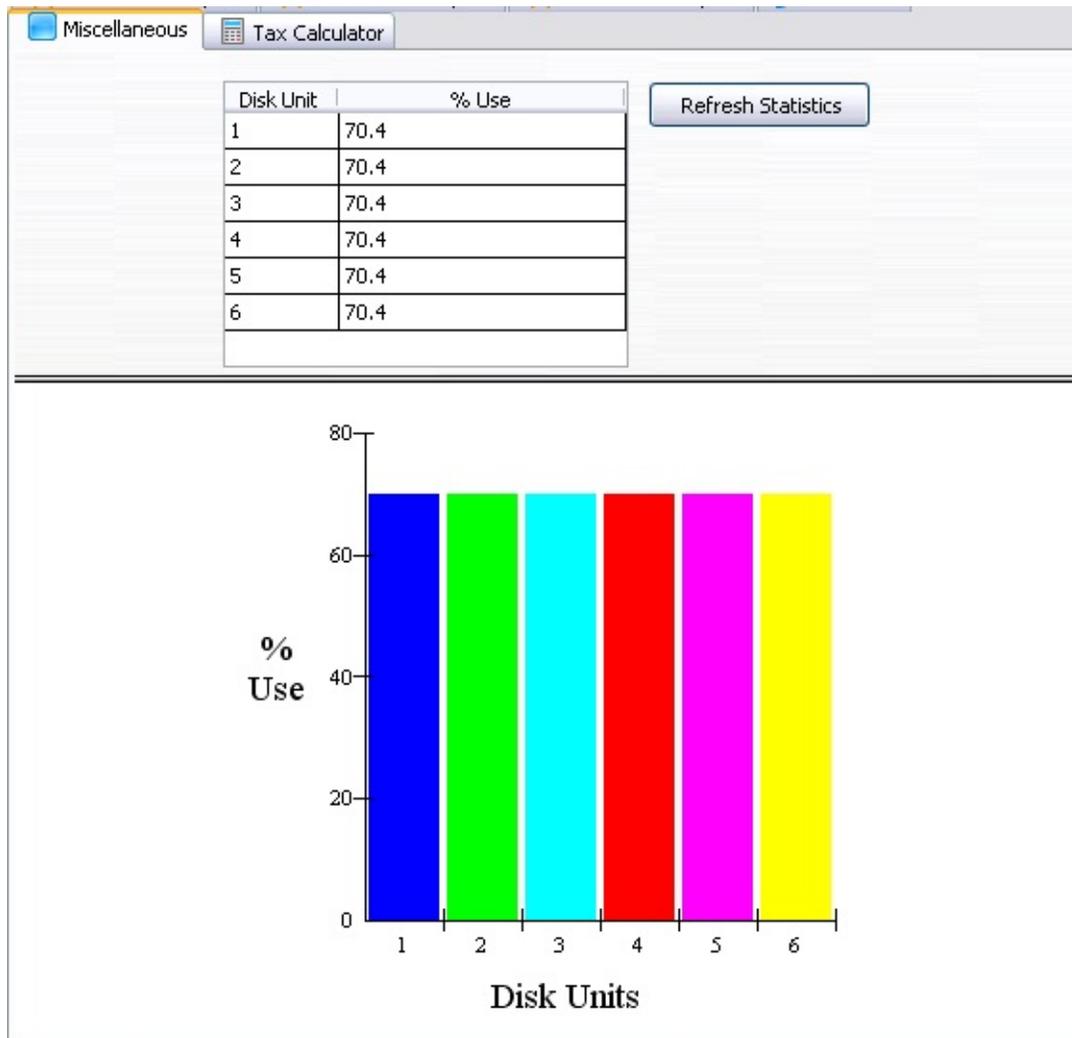
```
* Do any termination defined in the ancestor  
Invoke Method(#Com_Ancursor.uTerminate)
```

```
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 `wrkdisksts` 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.

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_ELTYTYP) 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_GRP_H) Name(#GRP_H_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_GRP_CL) Name(#GRP_CL_1) Columnrole(Label)
Displayposition(1) Parent(#GRP_H_1) Source(#VF_ELTYTYP)

Define_Com Class(#PRIM_GRP_CL) Name(#GRP_CL_2)
Columnsymbol(HollowUpTriangle) Displayposition(2) Parent(#GRP_H_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(181) 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)
```

```
*
```

```
=====
```

```
* This example shows a graph with the percentage of activity of each of the
System i disk units
```

```
*
```

```
=====
```

```
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_Ancessor.uExecute)
Invoke Method(#myscreen_wrapper.MakeRampTSAvailable)

Endroutine

Evtroutine Handling(#myscreen_wrapper.RampTSAvailable)

* Clear the lists
Clr_List Named(#DiskSts)
Clr_List Named(#grph_1)

* Run teh scripts to navigate to work with disk status
* The navigation only involves 3 screens: Login -> System i main menu, type
wrkdsts in teh command line -> Destination Work with disk status.

Invoke Method(#myscreen_wrapper.navigatetoscreen) Name(wrkdsksts)

Endroutine

Evtroutine Handling(#myscreen_wrapper.vHandleArrive)
Arrivedscreen(#CurrentScreen) PreviousScreen(#PreviousScreen)
Arrivedpayload(#Payload)

Define_Com Class(#prim_boln) Name(#MoreRecords)

Case (#CurrentScreen)

When Value_Is(= wrkdsksts)

```

```
* Payloads are destroyed when the ARRIVE script finishes executing.  
* The payload is set to FIRST_ARRIVAL when teh enter key is sent in  
System i main menu to differentiate what caused the screen to arrive:  
*   SETVALUE("cmdline","wrkdskst")  
*   SENDKEY(KeyEnter, "FIRST_ARRIVAL");  
* The same screen can arrive after sending the F10 or an F5, so we set  
payload to indicate so  
If (#Payload = RESTART_STATS)
```

```
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyF5)  
Payload(REFRESH_STATS)
```

```
Else
```

```
If (#Payload = FIRST_ARRIVAL)
```

```
#myscreen_wrapper.sendkey Key(#myscreen_wrapper.KeyF10)  
Payload(RESTART_STATS)
```

```
Else
```

```
#com_owner.uGetSubfilePage
```

```
Endif
```

```
Endif
```

```
Endcase
```

```
Set Com(#myscreen_wrapper) Visible(False)
```

```
Endroutine
```

```
* When RAMP-TS encounters screens like WRKDSKSTS or WRKACTJOB,  
it does not recognise the lists on them as subfiles (because they are not  
actually subfiles).
```

```
* Instead they are recognised as multiple fields.
```

```
* However, your scripts can still process them similarly to lists, as long as you  
follow a naming standard when naming the fields.
```

* In our WRKDSKSTS screen we identified the first column as UNITn and the fourth column as USEDn

Mthroutine Name(uGetSubfilePage)

#listcount := 1

Dowhile (#myscreen_wrapper.check_field_exists("USED" + #listcount.asstring)))

#myscreen_wrapper.getvalue From("UNIT" + #listcount.asstring)
Value(#vf_eltyp)

#myscreen_wrapper.getvalue From("USED" + #listcount.asstring)
Value(#vf_eltxts)

#VF_ELWIDP := #vf_eltxts.trim.asnumber

Add_Entry To_List(#DiskSts)

Add_Entry To_List(#grph_1)

#listcount += 1

Endwhile

Endroutine

Mthroutine Name(uTerminate) Options(*REDEFINE)

* Do any termination defined in the ancestor

Invoke Method(#Com_Ancessor.uTerminate)

Endroutine

* Refresh statistics

Evtroutine Handling(#PHBN_1.Click)

```
Invoke Method(#myscreen_wrapper.MakeRampAvailable)  
Foraction(RampTSAvailable)
```

```
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.

[Using Function Key Descriptions to Condition RAMP Buttons](#)

[Handling a Single Screen which Shows Multiple Modes](#)

[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](#)

Using Function Key Descriptions to Condition RAMP Buttons

This example shows how to match function key descriptions on 5250 screens:

```
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=Information Assistant  
F16=System Main menu
```

With RAMP buttons and function keys

This example uses JavaScript function [SHARED.apply5250FunctionKeyPatterns](#) which is designed to look for Fn=xxxxxx patterns on specified screen lines and then use these patterns to condition the RAMP-TS buttons and function keys to match.

Note that the matching does not include the text portion of the pattern. Normally in RAMP-TS the button captions are defined independently of the 5250 screen in RAMP-Tools.

To try out this example, copy and paste the functions in [SHARED.apply5250FunctionKeyPatterns](#) into your SHARED script object, file UF_SY420_RTS.JS. The logic in the functions is:

All RAMP-TS function keys and buttons are initially disabled.

The specified lines on the 5250 screen are searched for Fn=XXXXX patterns. The associated function keys and buttons are enabled by calling the standard SETKEYENABLED function.

If you have forced certain function keys to be always enabled this is then done via SETKEYENABLED.

If you have forced certain function keys to be always disabled this is then done via SETKEYENABLED.

Invoking the Function

Invoke the function from your destination screens arrival script - like this example:

```
vHandle_ARRIVE: function(oPayload, oPreviousForm)
{
  var bReturn = true;
  SHARED.apply5250FunctionKeyPatterns(22,23); /* Must be before
SHOW_CURRENT_FORM() */
  SHOW_CURRENT_FORM(true); /* Show the form in the framework and
show VLF buttons */
```

```

HIDE_5250_BUTTONS(); /* Hide any 5250 style buttons
displayed          */
GET_FORM_MESSAGE(22); /* Extract messages and hide the message
line              */
SETBUSY(false);     /* Last thing done - turn off the busy state */

```

This says to check on lines 22 and 23 of this arriving screen for Fn=xxxx text patterns and attempt to match the RAMP-TS buttons and function keys with them. Remember they are just text patterns and programmatically have nothing precisely to do with what function keys are actually enabled by the 5250 screen. The text patterns are put on the 5250 screens for humans to read - not computer programs.

Parameters

The function has these parameters:

Start Line Number	Mandatory. Integer. The first line to be searched for Fn=xxxx patterns.
End Line Number	Optional. Integer. The last line to be searched for Fn=xxxxx patterns. The default value is the same value as the start line number.
Keys/Buttons to always be enabled	Optional. Array of function key identifiers. For example [KeyEnter,KeyF11]
Keys /Buttons to always be disabled.	Optional. Array of function key identifiers. For example [KeyF12, KeyF3]

For example:

```

SHARED.apply5250FunctionKeyPatterns(22, 23, [KeyEnter,KeyF1],
[KeyF12] );

```

says to check screen lines 22 to 23, always enable Enter and F1, and always disable F12.

If you wanted to unconditionally enable the page up and down keys, which will not have matching text patterns to enable them you would do this:

```
SHARED.apply5250FunctionKeyPatterns(22, 23, [ KeyPageUp,  
KeyPageDown ] );
```

You could also do this permanently in the SHARED function.

Also see [Questions about the Function](#).

Questions about the Function

What should you do if SHARED.apply5250FunctionKeyPatterns does not work as expected?

Always try executing the screen with application level tracing turned on and examine the results.

Also check that extra buttons that always appear when executing in design mode (eg: Probe Screen) are not clouding the issue. The real test is to execute the logic in UF_EXEC (end-user) mode.

The destination screen's caption for the button may be clouding the issue. The text pattern on the 5250 screen might say F7=Hop, but the screen's definition in RAMP tools might say that the F7 key is to have a button that is captioned "Skip".

Can you change the SHARED.apply5250FunctionKeyPatterns logic?

Yes. This code is provided as an example for you to use in your SHARED object.

It is your code to change and maintain as you see fit.

For example, you might decide that you want to enable all the function keys initially. This is a possible solution when you have F24=More Keys style screens, because it is impossible to work out which function keys are actually enabled from the current screen only. To do this you would change this line:

```
for (var i = 0; i < oForm.vFKERTS.length; i++) { if  
(oForm.vFKERTS.charAt(i) != "0") oForm.vFKERTS =  
InsertString(oForm.vFKERTS,"0",i); }
```

To this:

```
for (var i = 0; i < oForm.vFKERTS.length; i++) { if  
(oForm.vFKERTS.charAt(i) != "1") oForm.vFKERTS =  
InsertString(oForm.vFKERTS,"1",i); }
```

You might decide that you always want to enable Enter and F1 without the caller always having to specify this - so you could add this to the end of SHARED.apply5250FunctionKeyPatterns:

```
SETKEYENABLED(GLOBAL_oCurrentTSform.symbolicName,KeyEnter,true,1)
SETKEYENABLED(GLOBAL_oCurrentTSform.symbolicName,KeyF1,true,1)
```

You might also decide to hook up the xxxxx text portion of Fn=xxxxx strings with the button caption in some way. This is also possible by using the second key match element fkeyMatch[2] and the standard RAMP-TS OVERRIDE_KEY_CAPTION_SCREEN function.

You can probably now see that we could not possibly cover of all the options and combinations automatically and efficiently by having specialized RAMP-Tools options. There would be so many check boxes and options that no one would understand what they meant or did. Using a generic scripted approach like this is simpler and it allows you to to tailor the approach to your exact needs.

What if you want to put SHARED.apply5250FunctionKeyPatterns into every arrival script?

You will have to add it to existing scripts individually - unless you are already calling something in the SHARED object that you can hook into.

You can also change your arrival script skeleton so that SHARED.apply5250FunctionKeyPatterns is automatically generated into all new scripts. Search for “Script Skeletons” in the RAMP-TS guide (lansa050.chm).

SHARED.apply5250FunctionKeyPatterns

This example JavaScript function is designed to look for Fn=xxxxxx patterns on specified screen lines. Copy and paste the attached functions into your SHARED script object, file UF_SY420_RTS.JS.

```
// Apply Fn=xxxxxx function key patterns to buttons and function keys
enabled on the current RAMP-TS screen

apply5250FunctionKeyPatterns :
function(iLowRow,iHighRow,aForceEnable,aForceDisable)
{
  if (GLOBAL_oAXESInterface == null) return; // No AXES interface
  if (GLOBAL_oCurrentTSform == null) return; // No current AXES form
  if (oGLOBAL_CurrentFORM == null) return; // No RAMP-TS
  definition for the form

  TRACE("SHARED.applyFunctionKeyPatterns started");

  if (iHighRow == null) iHighRow = iLowRow;    // default is same as
  low row

  var allkeys   = "";
  var typeOUTPUT =
GLOBAL_oAXESInterface.Element.TYPE_OUTPUT;
  var oForm     = oGLOBAL_CurrentFORM;

  // Disable all function keys and buttons to start with.
  // Note that the function keys (oForm.vFKERTS) and the buttons
(oForm.vFKEVLF) are BOTH disabled here

  TRACE("SHARED.applyFunctionKeyPatterns is disabling all function
  keys and all buttons");
  for (var i = 0; i < oForm.vFKEVLF.length; i++) { if
(oForm.vFKEVLF.charAt(i) != "0") oForm.vFKEVLF =
InsertString(oForm.vFKEVLF,"0",i); }
  for (var i = 0; i < oForm.vFKERTS.length; i++) { if
(oForm.vFKERTS.charAt(i) != "0") oForm.vFKERTS =
```

```

InsertString(oForm.vFKERTS,"0",i); }

    // Strip all output fields on the specified lines to create a long string of
function keys strings

    for (var iRow = iLowRow; iRow <= iHighRow; iRow++)
    {
        var aElement =
GLOBAL_oCurrentTSform.getElementsByRow(iRow);
        for (var i = 0; i < aElement.length; i++)
        {
            var oAXESElement = aElement[i];
            if (oAXESElement.type == typeOUTPUT) { allkeys += " " +
oAXESElement.getValue(); }
        }
    }

    // This RegExp looks for strings of the form F1=XXXX (where "F" can
be F, PF, FP, CF
    // or Cmd) XXXX can be a string of any length terminating at more than
one space,
    // the end of the line or another instance of "F1=" (thats the ?= look ahead
group).
    // All groups are forgotten (that's the ?:) except the function number and
the XXXX text.

    var reFKey = ^b(?:F|PF|FP|CF|Cmd)(\d+)[=-](.*?)(?=
(?:\b(?:F|PF|FP|CF|Cmd)\d+[=-])|s{2,}|$)/gi;

    var fkeyMatch = reFKey.exec(allkeys);
    while (fkeyMatch != null)
    {
        var key = "F" + TRIM_RIGHT(fkeyMatch[1]);
        // Note that the function key and the button are both being enabled here
SETKEYENABLED(GLOBAL_oCurrentTSform.symbolicName,key,true,true
        fkeyMatch = reFKey.exec(allkeys);
    }

```

```
// Enable any forced buttons. Note that the function key and the button are
// BOTH enabled

    if (aForceEnable != null)
    {
        TRACE("SHARED.applyFunctionKeyPatterns is forcing the
enablement of specified keys/buttons");
        for (var i = 0; i < aForceEnable.length; i++) {
SETKEYENABLED(GLOBAL_oCurrentTSform.symbolicName,aForceEnabl
}
        }

// Disable any forced buttons. Note that the function key and the button
// are BOTH disabled

    if (aForceDisable != null)
    {
        TRACE("SHARED.applyFunctionKeyPatterns is forcing the
disablement of specified keys/buttons");
        for (var i = 0; i < aForceDisable.length; i++) {
SETKEYENABLED(GLOBAL_oCurrentTSform.symbolicName,aForceDisab
}
        }

// Finished

    TRACE("SHARED.applyFunctionKeyPatterns ended");

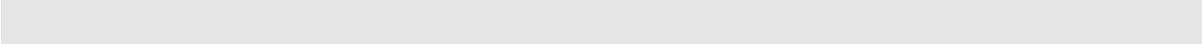
}, // <--- Note the comma
```

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.

See tutorial [RAMP-TS008: Changing Inquiry Screen to Update Mode](#) for an example of how to handle this.



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

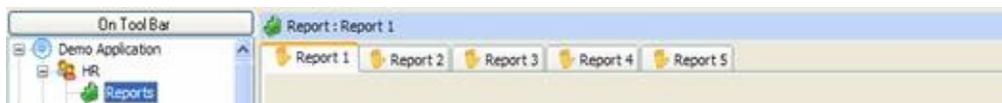
Each report is defined as a unique business object named "Report 1" through "Report 5". In this case the application navigation tree might be structured like this ...



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:

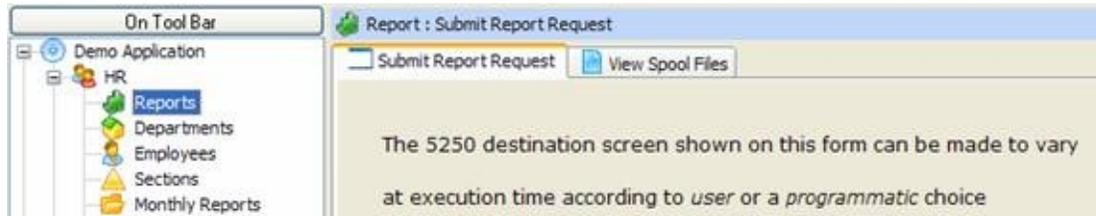
The first approach consumes too many business objects

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)
```

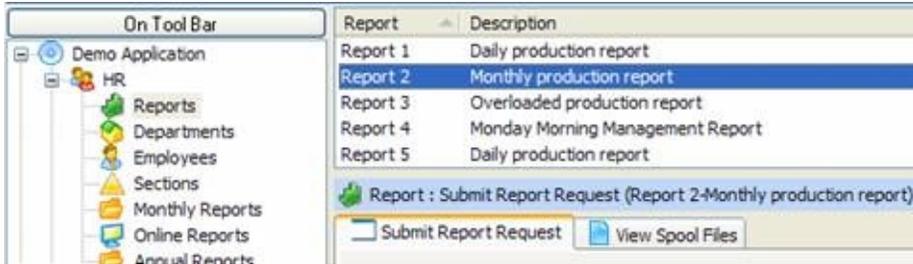
```
* Instance list updating has been completed
```

```
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 script that navigates to uReport1 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_SCREEN("uReport2"); return;
case 3: NAVIGATE_TO_SCREEN("uReport3"); return;
case 4: NAVIGATE_TO_SCREEN("uReport4"); return;
case 5: NAVIGATE_TO_SCREEN("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 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_SCREEN(objListManager.AKey1[0]);

    return;
}
/* Normal navigation logic to handle this screen */
NAVIGATE_TO_JUNCTION("whatever");

Etc, etc .....
```

Using this Approach in other Situations

This 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 could use this approach to cause three different 5250 destination screens to be displayed on a single command handler tab named "Details".

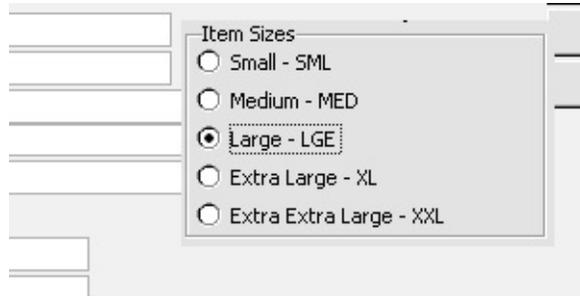
That the instance list can be used to dynamically create a "menu" of 5250 destination forms.

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_SCREEN() 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 set 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:

	5250 Field Name	Function Key	VL Handler (class VF_AC017 object)
1	txtSTATE	F4	P_STATE
2	txtPHONE	F5	P_PHONE
3		F4	
4		F4	

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.

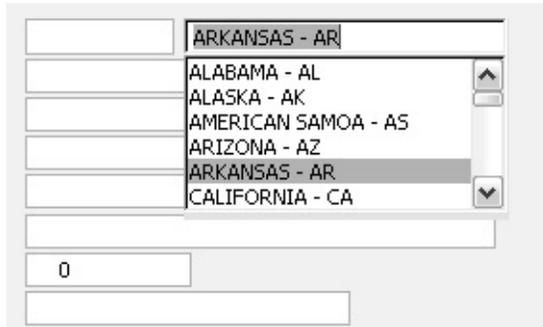
When the user makes a selection, the VL form can alter the value of any named field on the current 5250 destination form.

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:

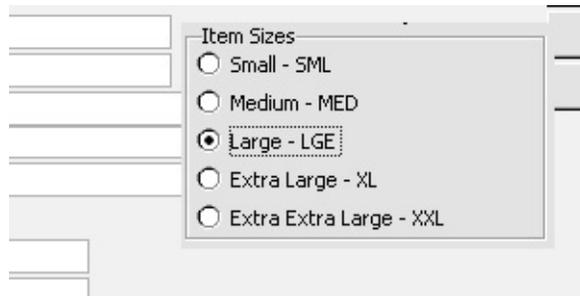
Combo Box

DF_PRM01 prompts using a combo box of US states like this:



Radio Buttons

DF_PRM02 prompts using a set of product size radio buttons:



List with Columns

DF_PRM03 generically prompts for employees by name:

▾
 ▾
 ...

DF_P52503 - Prompting Example 3

Employee Surname

Employee Number	Employee Surname	Employee Given ...	Post / Zip Code
A1031	BLAKE	JOHN	0
A0090	BLOGGS	FRED JOHN	0
A3564	BROWN	FREDDY	0
A0070	BROWN	VERONICA	0

Tree

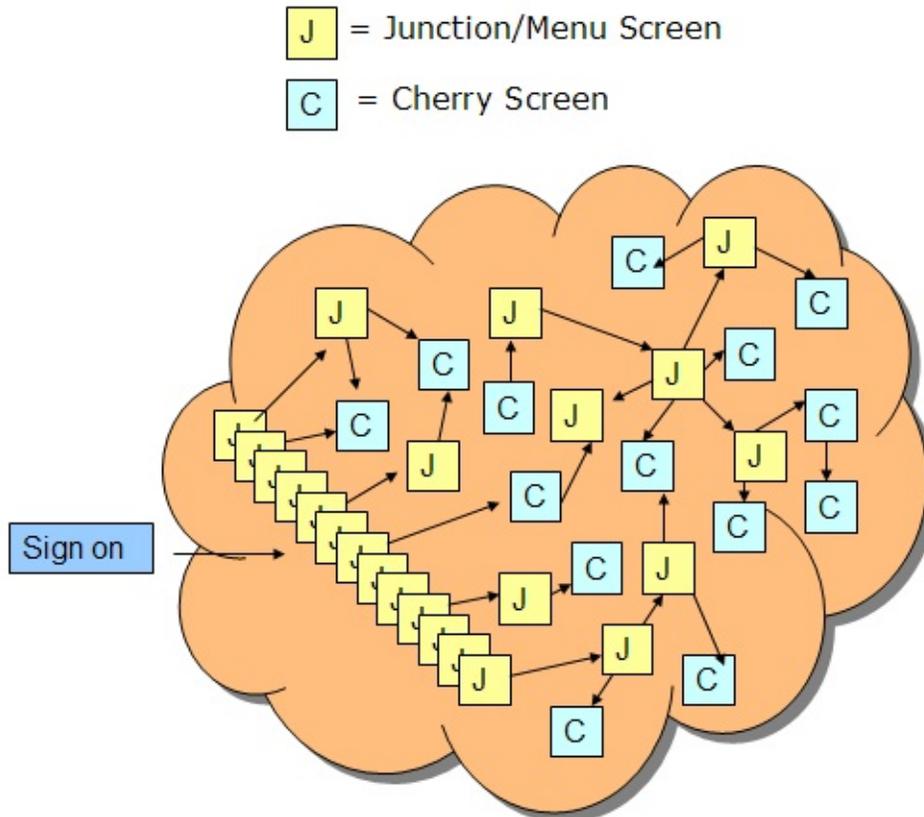
DF_PRM04 prompts department and section information using a tree:

▾
 ▾

- # ADMINISTRATOR DEPT
- # INTERNAL AUDITING
- # FLEET ADMINISTRATION
- # GROUP ACCOUNTS
- # INFORMATION SERVICES
 - ... ADMINISTRATION
 - ... **PURCHASING**
 - ... ACCOUNTING
 - ... DEVELOPMENT
- # LEGAL DEPARTMENT
- # MANAGEMNT INFORMATIO

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:

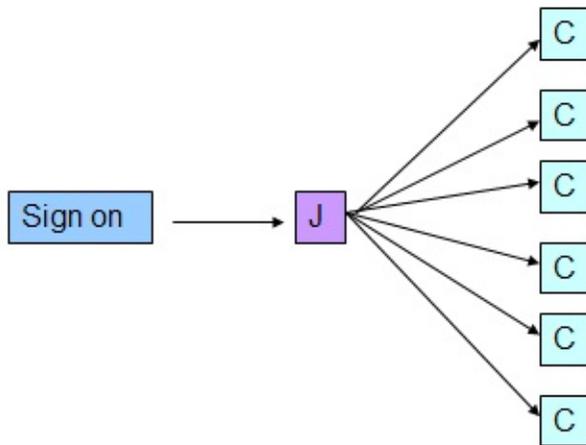


A 5250 user signs on and navigates around a cloud of menus/junctions to reach the "cherries" (5250 destination screens) where they do useful work.

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.

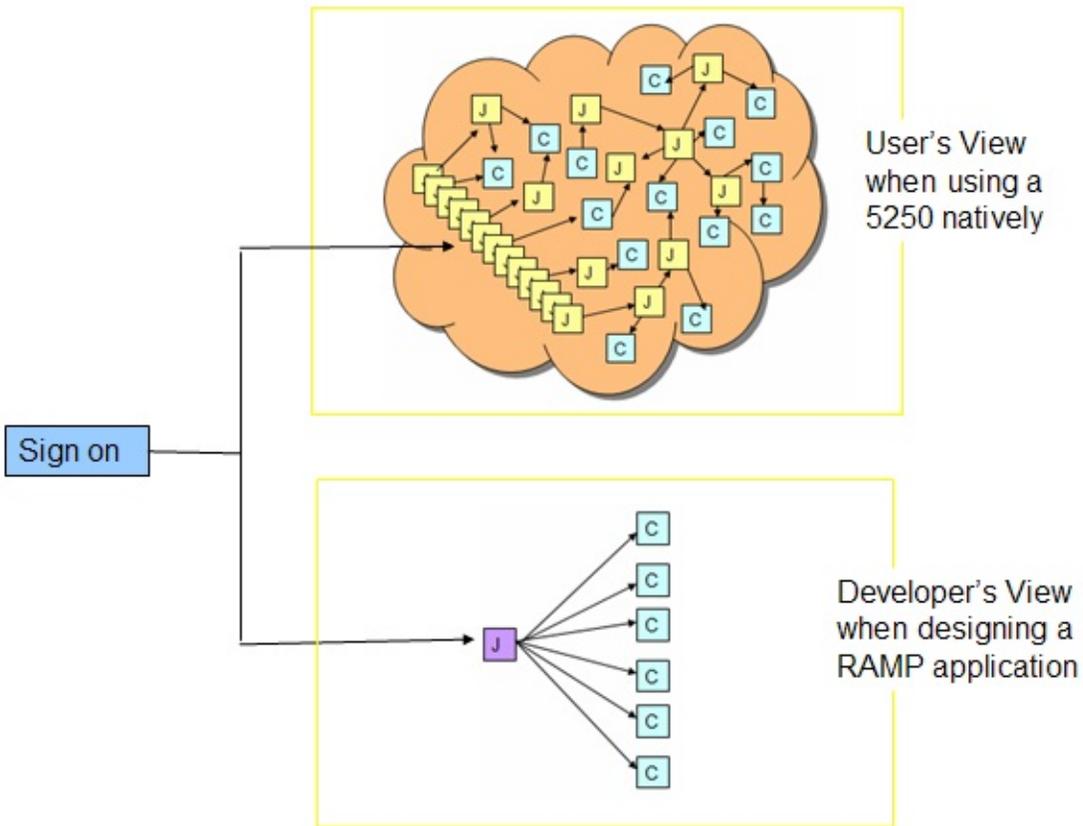
The invocation scripts for the destination screens are simpler and standardized.

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.

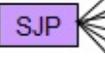
The SJP model cannot be applied to every type of application, but where it can be applied it may represent a saving in the time taken to develop a RAMP application.

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  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

 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:

RAMP Destination Invoke Script executing on Client

```
/* Call program CUSTINQ */
NAVIGATE_TO_JUNCTION("SJP");
SETVALUE("PGMNAME", "CUSTINQ");
SENDKEY(KeyEnter);

/* Now get CUSTINQ to display the customer */
SETVALUE("CustomerNumber", "123456");
SENDKEY(KeyEnter);
```

SJP Program (CL) executing on iSeries

```
/* Wait for instructions from client */
L10: SDRVF RCDFMT(contains the field &PGMNAME)
/* Call the specified program */
CALL PGM(&PGMNAME)
/* Go back and wait for another request from client */
GOTO L10
```

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 display 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 returned 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 as RPG, COBOL, C or RDML (which is really RPG anyway).

If you have LANSAs programs RDML is a good choice because it makes it very easy to call LANSAs 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 scripts

Here the 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 */

<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 in the script 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

xxxxxxx 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 executing RAMP applications:

Strange behavior in scripts

Screen does not react when selection is changed in instance list

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 FORMSCRIPT_137.

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.

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 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:

See the topic [Interacting with Instance Lists in Scripts](#)

See the topic [Replacing Hardcoded Employee Number with Current Instance List Entry](#).

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 in 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_nnnnnnnnnnnnnnnnn.js are generated, one for each session where the nnnnnnnnnnnnnnnnn 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).

Frequently Asked Questions

How is a Framework associated with RAMP-TS or RAMP-NL?

I have made a mistake in classifying a screen. How do I change the classification?

How do I differentiate two screens which have the same name?

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 should the F12=Cancel and F3=Exit buttons and function keys be disabled on every 5250 screen?

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?

Can you add in your own functions to the scripts?

How do I make my scripts work in multiple partitions?

How to tell in a RAMP-TS script what theme is being used?

How do I stop break messages in aXes 5250 sessions?

How to tell in a RAMP-TS script what theme is being used?

In a RAMP-TS script you may want to know what VL/VLF theme is being used in the application.

The JavaScript variable GLOBAL_VL_Theme is accessible to RAMP-TS scripts. It should contain “BLUE”, “SILVER”, “OLIVE” or “GRAPHITE”.

You should not change the content of GLOBAL_VL_Theme.

How is a Framework associated with RAMP-TS or RAMP-NL?

A framework (as defined in its XML file) is specifically bound to either RAMP-TS or RAMP-NL. This means it can only be used with that type of RAMP session.

When you first start to use RAMP with a framework it makes a decision on whether to bind to RAMP-NL or RAMP-TS. The binding process works like this:

First, the content of any existing RAMP nodes file, as defined on the (Framework) -> (Properties...) à Framework Details tab as the Nodes XML File is examined. If the nodes file already contains RAMP-TS or RAMP-NL destinations and scripts, the framework is automatically bound to RAMP-TS or RAMP-NL as appropriate.

If the Nodes XML File is empty or non-existent, then the servers associated with the framework are examined. If one or more RAMP-NL servers are defined and no RAMP-TS servers are defined, the framework is automatically bound to RAMP-NL.

In all other situations the framework is automatically bound to RAMP-TS.

When you start RAMP Tools the window title shows whether the framework is bound to RAMP-TS or RAMP-NL.

Things to watch out for when binding a framework to RAMP-TS or RAMP-NL are:

Make sure the servers associated with a framework are correct and not mixed. Normally you would only have RAMP-TS or RAMP-NL server(s) defined, but not both. If necessary create different server XML files for different frameworks to avoid having RAMP-TS and RAMP-NL servers mixed within one framework.

When using Save As to create a new framework, also change the Nodes XML file to a new file name. After saving the new framework, delete the Nodes XML file so as to start working on the new framework with a new and empty nodes file.

If you accidentally bind a framework incorrectly to RAMP-NL or RAMP-TS, open the framework XML file with NOTEPAD and search for this property definition `<PROPERTY NAME="U5250HANDLER" VALUE="TS"/>` or `<PROPERTY NAME="U5250HANDLER" VALUE="NL"/>`.

Remove this property from the XML file and save the changes. Start the

framework as a designer and go to the (Framework) -> (Properties...) à Framework Details tab. Check that the **Nodes XML File** name is correct and different to any other framework you have. Check that you have a either RAMP-TS or RAMP-NL server(s) defined. Do not have both types defined. Start the framework as a designer and then start RAMP Tools again. The binding logic defined previously will be performed again when you start RAMP Tools.

If the binding process gets the wrong results again it is because your framework's servers are not correctly defined, or, because your Node XML file contains definitions that force it to bind to RAMP-TS or RAMP-NL.

I have made a mistake in classifying a screen. How do I change the classification?

Delete the screen definition in the 5250 [Screen and Script List](#). The screen will appear as undefined in the Tracking Information area.

Remember that the scripts associated with the screen are also deleted, so you need to retrack the navigation from and to the screen.

How do I differentiate two screens which have the same name?

Use the Variant Name to differentiate screens that were given the same name. See [Your RAMP-TS Screen Script Defines a JavaScript Object](#).

The Variant Name is a property of the screen in the same way as the vName, vType, etc. To reference the Variant Name in your script use:

```
this.vLatestVariant
```

Typically you would use the Variant Name in screens that have different views. A typical example would be a screen like Work with Active Jobs. You could name it WRKACTJOB. When you press F11 the same screen appears but this time with a different set of columns.

You can use one of the column headings in this screen that was not present in the previous one as part of the screen definition and save it using the Variant Name. Likewise, pressing F11 again gives you another different set of columns. You can again use one of the column headings in the definition and save the screen with another variant name.

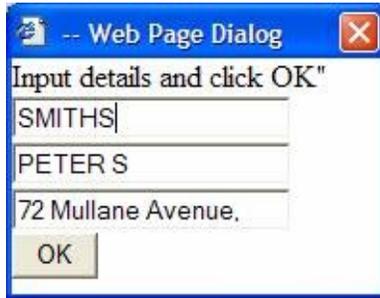
For more information see [RAMP-TS015: Understanding and Handling Screen Variations](#).

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>
</HEAD>
<BODY onload="BODY_Load();" onunload="BODY_UnLoad();" >
<script>
function BODY_Load() /* Map arguments passed in to web form fields */
{
    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();">
</BODY>
</HTML>
```

It looks like this when displayed:



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;dia
      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 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.

Do I have to identify and script every 5250 screen in my application to modernize it?

No.

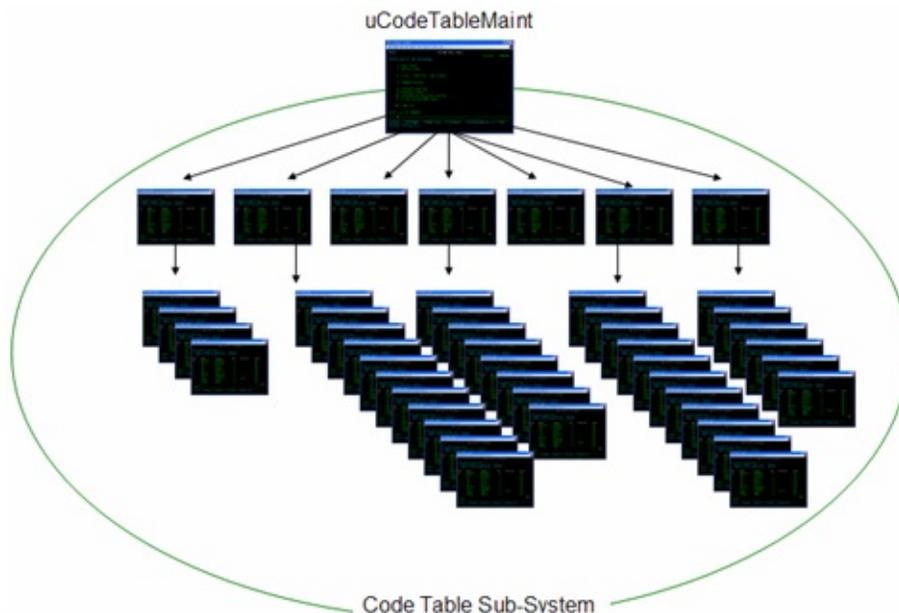
Typically some areas of a 5250 application are rarely used or used by very few users.

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` 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:

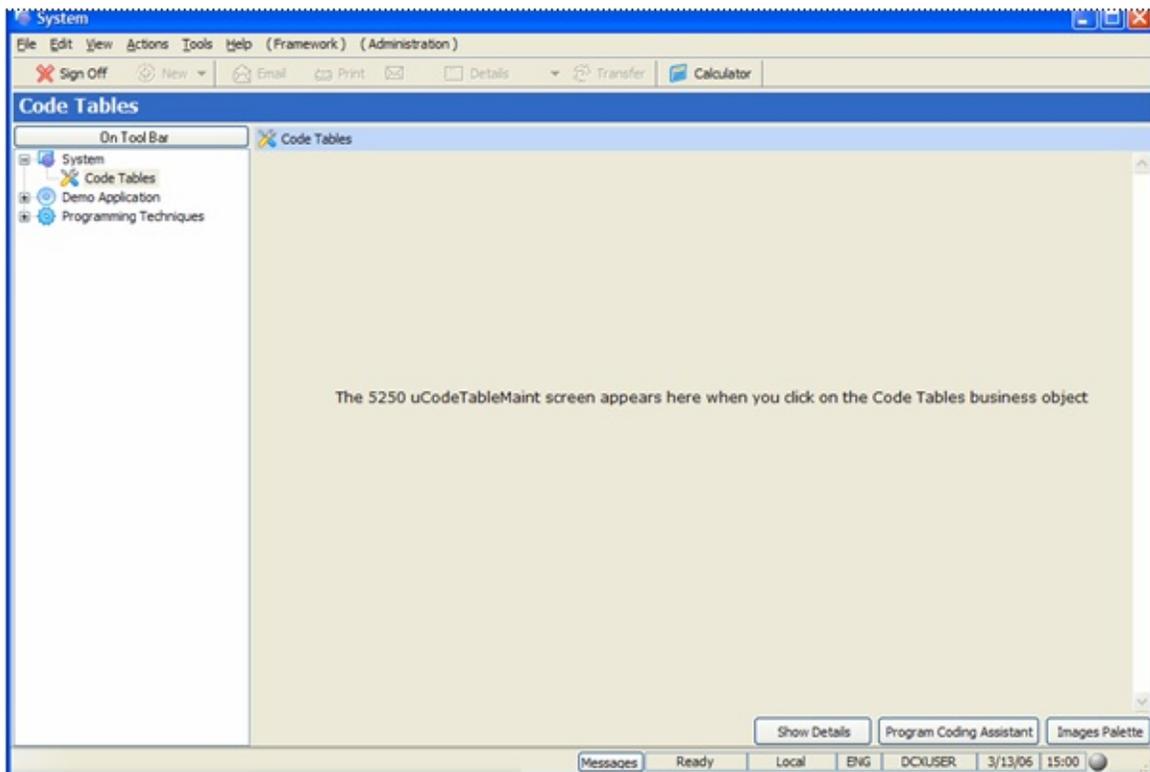
Create a business object called Code Tables and associate with an application.

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.

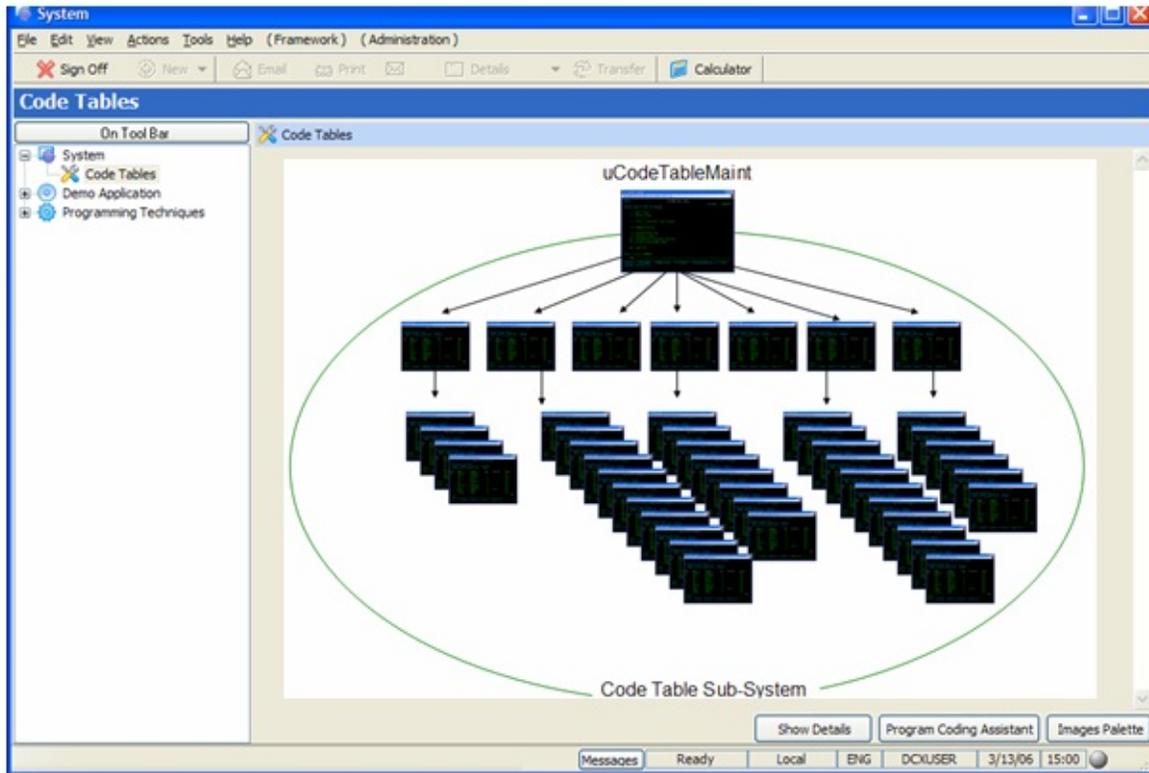
Identify and define the 5250 work with screen uCodeTableMaint to the Framework as a destination screen and associate it with the Maintain screen.

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 [Scripting Pop-up Menu](#) : choose Current Framework and then partition.

Can you add in your own functions to the scripts?

Yes, you can add a function to your screen script. This means it is hidden from all other screens and only accessible to the screen object itself.

The format is

```
MyFunction : function (parms) { logic } ,
```

And it is invoked by

```
this.MyFunction(parms);
```

Put the function before the <SYSINFO> block. Note trailing comma.

Alternatively, you can add a function to UF_VFSY40_RTS,JS and then is accessible to all screen objects. By passing your "this" reference to the function, it can generically act on behalf your behalf. See [User-Defined Script Functions](#).

How do I stop break messages in aXes 5250 sessions?

aXes does not like IBM i break messages - because they are pushed by the server – something that a pull based technology like the web browser cannot accommodate.

It is strongly recommended that you don't use break message in aXes 5250 sessions - especially in RAMP-TS and aXes-Mobile applications.

Using these IBM i CL commands in a user's IBM i logon program will generally stop all break messages:

```
CHGJOB BRKMSG(*HOLD) STSMMSG(*NONE) MONMSG  
MSGID(CPF0000 MCH0000) CHGMSGQ  
MSGQ(*WRKSTN) DLVRY(*HOLD) MONMSG MSGID(CPF0000  
MCH0000)
```

```
CHGMSGQ MSGQ(*USRPRF) DLVRY(*HOLD) MONMSG  
MSGID(CPF0000 MCH0000)
```

Appendix

Where and how are my RAMP-TS screen identifications kept?

By default RAMP-TS stores your screens in a folder named `/ts/screens`.

Each screen definition is stored in a file named `N.scn` (eg: `1.scn`, `2.scn`, `3.scn`, etc). They contain JSON fragments that define the characteristics of the screen.

These files are only required during development. For run time, all these of these files are published in a single consolidated file named `screens.jsn`.

`Screen.jsn` is the only screen file that needs to be deployed to production environments

Important things to know about this folder

Back it (and any subfolders) up regularly. The content of the `N.scn` files represents a significant investment of your time.

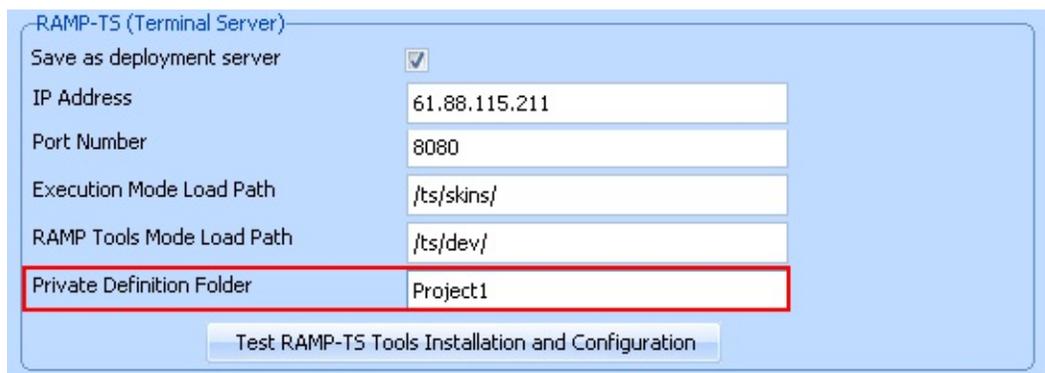
Don't copy from, copy to, move, rename or delete any file in folder `/ts/Screens`, or any subfolder of it.

Only ever change the content of these files via the RAMP-TS editor.

Dividing up screen definitions

Your screen definitions do not have to be in `/ts/screens`, you can divide up screen definitions into sub-folders like `/s/screens/Project1` or `/ts/screens/Project2`.

You just input the folder name (`Project1`) with nothing else. It is implicit that this is a sub-folder of `/ts/screens`:



The image shows a configuration window titled "RAMP-TS (Terminal Server)". It contains several input fields and a checkbox. The "Private Definition Folder" field is highlighted with a red border and contains the text "Project1".

Save as deployment server	<input checked="" type="checkbox"/>
IP Address	61.88.115.211
Port Number	8080
Execution Mode Load Path	/ts/skins/
RAMP Tools Mode Load Path	/ts/dev/
Private Definition Folder	Project1

Test RAMP-TS Tools Installation and Configuration

When you define a RAMP-TS server in the framework, this subfolder name is specified as a private working set. Screen definitions may be divided up like this on an indivisible and completely independent project basis.

Screens should never be divided up on a unit of work or developer basis because you must not copy, move, rename or delete them.