e-SIM MMI Reference Design 1.x

Training

Version 2

Table of Contents

e-SIM Ltd.
POB 45002
Jerusalem 91450
Israel

Tel: 972-2-5870770
Fax: 972-2-5870773

Written and produced by e-SIM Ltd.
Document Version 2.0

e-SIM MMI Reference Design 1.x **Training**

RapidPLUS version: 8.0
E-SIM MMI version: 1.2
Training Book version: 2.0
Feedback: support@e-sim.com

Course Objectives

- To explore and explain the e-SIM MMI Reference Design solution
- To teach you how to launch a project using the e-SIM MMI
- To fit the training to your specific needs

Agenda

- Architecture of the e-SIM MMI Reference Design
- Customization tools
- Main components overview
- Implementation guidelines
- Synchronization of components
- Add new MMI module – Final project

Syllabus: Day 1/5

Architecture	
9:00	Course Orientation
9:00-9:15	(1) Introduction to e-SIM MMI
10:00-12:30	(2) Architecture
12:30	Lunch
13:30-15:30	Architecture
15:30-17:00	(3) Main components

esim

Syllabus: Day 2/5

Architecture Exercises	
9:00-9:15	Summary of Day 1
9:15-11:30	Basic architecture exercise
12:30	Lunch
13:30-17:00	Basic architecture exercise

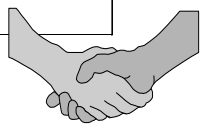
esim

Syllabus: Day 3/5

Customization Tools	
9:00-9:30	Architecture exercise summary
9:30-10:30	(4) Implementation guidelines
11:15-12:00	(5) Customization tools – Layout Manager
12:00-12:30	(5) Customization tools – Menu and Text Generator
12:30	Lunch
13:30-14:00	(5) Customization tools – Multimedia Resource Manager
14:00-15:00	Customization tools – Exercise
15:10-17:00	(6) Synchronization of components

Syllabus: Day 5/5

Final Exercise	
9:00-9:15	Summary of Day 3
9:15-9:30	(9) Final exercise - review
9:30-12:30	Final exercise
12:30	Lunch
13:30-16:30	Final exercise
16:30-17:00	Training summery

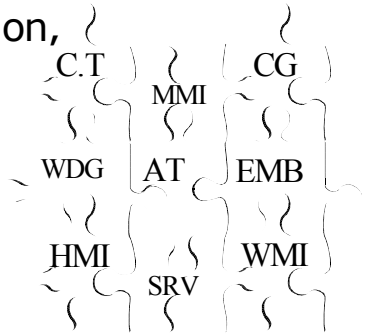


Syllabus: Day 4/5

Advanced Methods	
9:00 - 9:15	Summary of Day 3
9:30 - 10:00	(7) Draw method
10:15 - 12:30	Draw Exercise
12:30	Lunch
13:30 - 14:00	(8) Menu method
14:10 - 17:00	Menu exercise

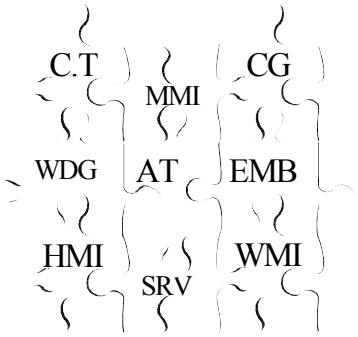
Overview

- MMI (man-machine interface) reference design for a mobile telephone
- Provides a complete solution, including the platform, MMI application, and RapidPLUS tools that can customize the MMI application



Overview (cont.)

- Can be easily customized to fit the needs of other platforms and projects
- The modular design also allows several developers to work on the project simultaneously



Vocabulary

- HMI ~ MMI
- MMI modules
- Widgets
- Services
- Embedded components
- Customization tools

Where to Get More Information

- Personal support from an e-SIM engineer
- Software Design Document
- Customization tools guides documents (3 guides)

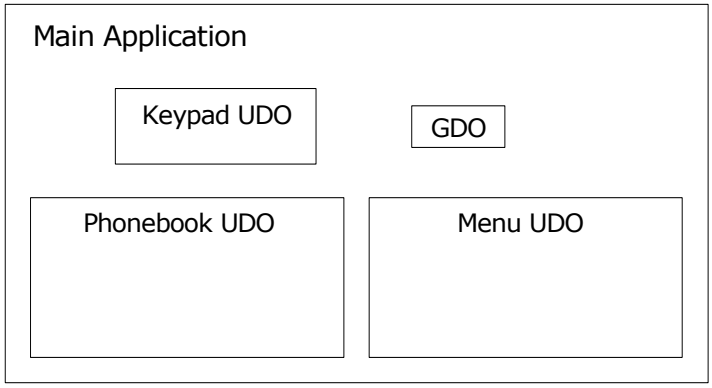
e-SIM MMI
Reference Design 1.x
Architecture

Component Types

MMI	Man Machine Interface Modules (UI) referred to as HMI and WMI modules
WDG	Widgets (activated on request)
SRV	Services (Respond at any time)
EMB	Embedded (hardware/simulation)

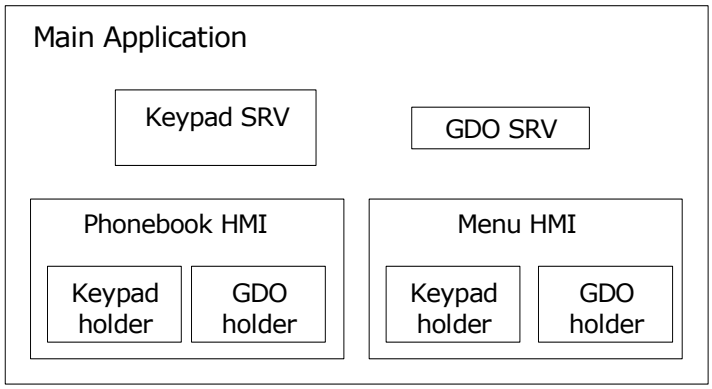
Flat Structure

- User objects used for embedded components only



Complex Component Structure

- User objects used to split the application logic into sub-components



esim

Embedded Components

Low-level system components such as keypad, display, memory, audio, battery.

They:

- Have the graphics of the product for input and output devices and for simulation panels
- Define the API to the embedded elements
- Include the minimum code that uses the object interface

EMB

esim

MMI Modules (user interface)

High-level components such as the phone book interface and call management.

They:

- Describe the main features of the product
- Handle the MMI: screens, menus, texts, and soft keys
- Are activated by the user

MMI

EMB

MMI Modules (user interface) (cont.)

There are two types of MMI modules:

- WMI modules, at the lower level, are reusable MMI components that can control widgets, services, and embedded components
- HMI modules, handle a subsection of the project MMI. They can control all of the components, plus the WMI modules

Service Components

Middle-level components such as components that control the keypad and display.

They:

- Manage the project's data and state
- Act as buffers between embedded components and MMI modules
- Can respond to requests at any time because they are always available

Widget Components

Middle-level components such as the scroll bar.

They:

- Provide a single unit of user interface functionality
- Are activated on request and deactivated when no longer needed
- Have no self memory and must be set with data whenever activated
- Reusable

Main Application

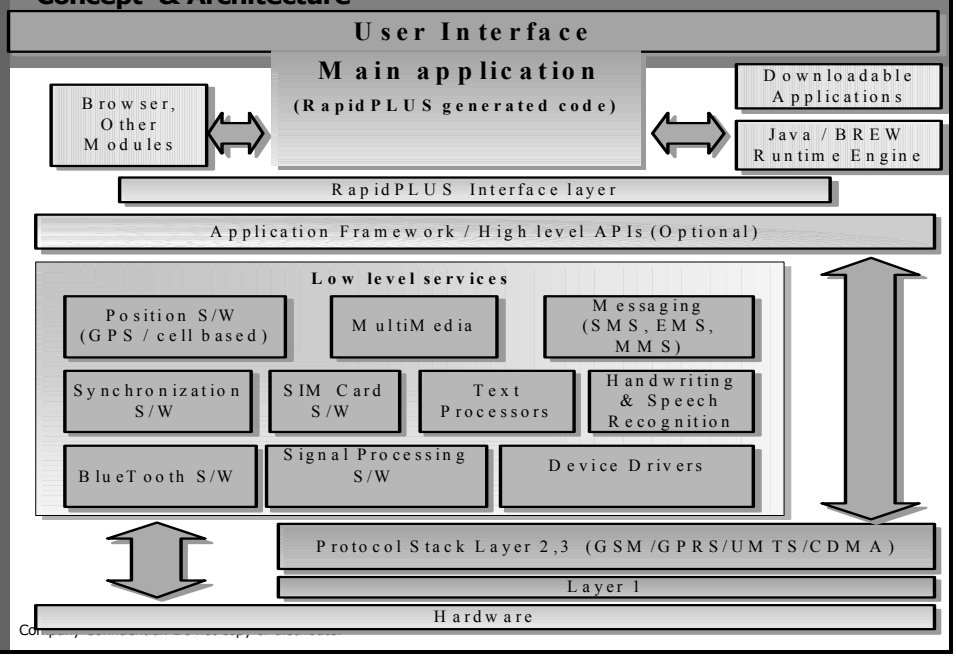
- Manages the entire project by activating the various components according to priorities.
- Is the umbrella application that contains all the other components.

Component Types Summary

MMI	WDG	SRV	EMB
Idle	Editor	Calls	Keypad
Call	Animator	Display	Battery
Menu	List	Bitmaps	Audio
SMS	Scroll bar	Call records	Memory

Main features	Parent activated	Always available	API as in embedded
User activated	Re-usable	Have memory	Have graphic objects
	Have no memory	Manage data	Minimum code

Concept & Architecture



Device Arbitration (conflict handling)

Version 1.x:

- Main application manages resources
- One MMI module is active at any time
- HMI may start other HMI and give it the resources
- Each MMI must implement Init, Start, Stop, Pause, Resume

Power Saving

Rapid application is event driven:

- No infinite loops or polling
- MMI is idle most of the time; activated upon external events
- The State machine works in cycles:
  - Run idle → moreToDo
  - Prevent system starvation even in simple operating system

Power Saving (cont.)

Timer on request API:

- Timers are handles by the RapidPLUS state machine (kernel)
- Kernel API provides 2 callback functions (start/stop timer)
- The system calls a RapidPLUS function when the timer expires

Factory Customization Concept

All resources are stored in external files:

- Bitmaps and animation bitmaps
- String literals
- Menu indexes
- Phone settings
- Layout data
- Fonts

Factory Customization Concept (cont.)

- RapidPLUS application only needs the array entry point for each resource
- Different configurations can be prepared; no need for code generation

Memory

- ROM
  - State machine engine; size varies between 30 to 50 KB.
  - Graphic library; size ~15 KB.
  - Application logic and objects
  - Constant objects

Total~1 MB

Memory (cont.)

- RAM
  - State machine queues and messages; for an empty application this represents 300 KBytes
  - Variable objects
  - (Cellular phones today has 300 KB RAM for the simplest and up to 2 MB for the advanced phones)

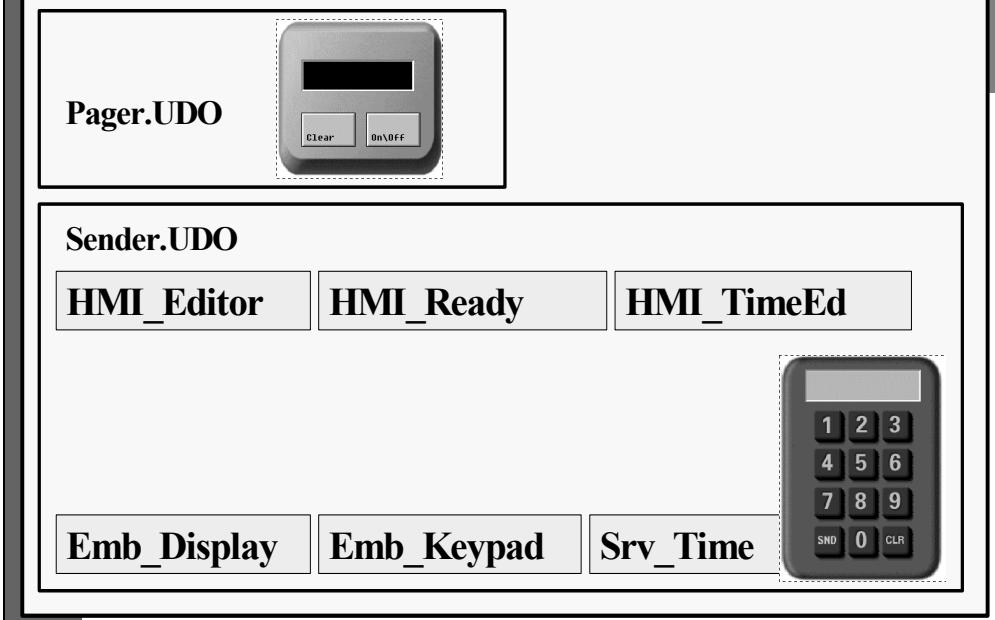
Documentation

- Each of the customization tools is described in its own document
- The SDD document describes the design, implementation, and customization of the Reference Design. It also provides details about each of the components.
- Training books
- All Given in the GivenResource folder

Architecture Exercise - Pager

- Using holders, DMA
- One HMI active at a time
- Follow the architecture slide
- PagerSys.rpd

Main_App.RPD



Notes:

e-SIM MMI
Reference Design 1.x

Main Components Overview

Component Types

- MMI modules
- Service components
- Widgets
- Embedded interface components

MMI Modules

- **HMI_IDLE.UDO**
  - Handles all activity of the MMI when the phone is idling
  - Serves as a junction from which other MMI modules can be operated, depending on the inputs that occur
- **HMI_CALL.UDO**
  - Handles phone call interaction: outgoing calls, incoming calls, multi-party calls, dialing, hold, speed dialing, and voice-mail

MMI Modules (cont.)

- **HMI_PBOOK.UDO**
  - Handles phone book interaction and high-level functionality
  - High-level functionality includes phone book selection, copying between phone books, selecting from the list of entries, managing speed dial keys, and memory used
- **HMI_MENU.UDO**
  - Performs or initiates much of the function-ality associated with the menu items

MMI Modules (cont.)

- **WMI_MENU.UDO**
  - Handles menu navigation
  - Used by HMI_MENU and by other MMI modules when menu display is needed. For example, call options and Contacts menu
- **WMI_POPUP.UDO**
  - Handles different types of pop-up messages (timed out and non-timed out)

Services: Modem Services

- **SRV_CALL.UDO**
  - Manages the active calls
  - Holds data of all active calls and manages call manipulation, such as calls-on-hold and multi-party calls
  - Uses SRV_TAPI.UDO for low-level functions
- **SRV_TAPI.UDO**
  - Manages telephony functions, including call manipulation, network services, and certain SIM services

Services: Modem Services (cont.)

- **SRV_AT.UDO**
  - Known as the AT Command Dispatcher, manages all traffic to and from the modem
  - Provides basic parsing of the AT commands that are sent from the modem
  - Provides a simple synchronization mechanism to all users of the services to wait and resend their requests when the modem is busy

Services: Display Services

- **SRV_LAYMAN.UDO**
  - Referred to as the Layout Manager
  - Knows what to draw and where, for a given layout
    - Layouts are defined using the Layout Editor tool
    - A layout specifies the position of widgets, text, and bitmaps
    - Only one layout is active at any time

Services: Display Services (cont.)

- **SRV_PLOTTER.UDO**
  - Provides the glue between the MMI and the layout
  - When a layout is initialized, it tells the plotter about all of its primitive elements
  - Application draws these elements by calling plotter functions and specifying the layout element ID as a parameter

Services: Display Services (cont.)

- **SRV_DISPLAY.UDO**
  - Provides all the drawing functionality required by the application
  - All display changes ultimately are performed through this component
  - Usually used via the plotter

Services: Display Services (cont.)

- **SRV_BITMAPS.UDO**
  - Project bitmaps and animations are managed using the Multimedia Resource Manager
  - Graphic files can be imported into an object array in SRV_BITMAPS.UDO
  - IDS can be imported into a constant array in RPD_CONSTANTS.UDO

Widgets

- **WDG_EDIT_BOX**
  - Graphic widget for the editor
  - Draws text on the display according to the selected orientation
  - Draws the selected cursor in the correct place in the text
- **WDG_LIST.UDO**
  - Handles list look-and-feel and navigation

Widgets (cont.)

- **WDG_SCROLLBAR.UDO**
  - Provides scrolling functionality for menus and lists
- **WDG_SOFTKEY.UDO**
  - Handles the two soft keys
  - Draws the soft key labels
  - Triggers exported events when the keys are pressed

Embedded Interface Components

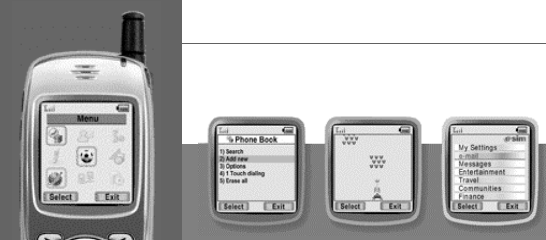
- Always generated as "interface only"
- Objects and internal logic are not generated
- **EMB__GSRV.UDO** (general services)
  - Contains interfaces to audio, battery, backlight, microphone, clock, registry, vibrator, and ear-bud
  - Provides functions for writing to and reading from persistent (flash) memory
  - Simulation panel allows the simulation user to modify the settings of the various devices

Embedded Interface Components (cont.)

- **EMB_KPD.UDO** (keypad)
  - Contains the interface between the hardware and the keypad
  - Nested in SRV_KEYPAD.UDO, which provides a more convenient interface to the MMI
- **EMB_AT.UDO** (modem)
  - Responsible for communication with the modem

Miscellaneous Component

- **TRACE.UDO**
  - This component contains assert functions



e-SIM MMI
Reference Design 1.x
Implementation Guidelines

Contents

- Naming conventions for modes, objects, and logic
- Required basic modes and interfaces

Mode Names

- Each word is capitalized
- No underscores between words

Example: IntializationProcess

User Object Names

- Prefix indicates the object type:
  - EMB for embedded components
  - SRV for service components
  - WDG for widget
  - HMI or WMI for MMI modules
- Prefix followed by an underscore

User Object Names (cont.)

- Object name in uppercase
  - Two or more words are separated by an underscore
  - When an instance of a user object is added to a parent application, its UDO name is used

Example: SRV_DISPLAY

Object Names

- Each word is capitalized
- Name followed an underscore, followed by the object type:

- Ary for array
- Num for number
- Bmp for bitmap
- Pb for pushbutton
- Ev for event
- Str for string
- Img for image
- Tmr for timer
- Int for integer
- Tik for timer tick

Example:
Send_Pb

Constant Set Names (cont.)

- Constant **item** names in uppercase
- Underscores separate each word

Examples:

FUNCTION_ID_Cs (a constant set)
CALL_WAITING (a constant set item)

User Function Names

- Capitalization used to separate adjacent words

Example: drawText:onElement:

Exported Event and Property Names

- Each word is capitalized
- Name followed an underscore, followed by the event/property type

Example: Selected_Int

Required Modes for MMI Modules

- Basic mode tree for each MMI module contains these modes:
  - **Idle:** component is awaiting activation
  - **Active:** contains entire functionality of the active component
  - **Pause:** all activity ceases. This mode is exited when the component is either resumed or stopped
  - **Stop:**

Required Functions & Events

- Initialization User Function (init)
  - Initializes the MMI module's holders
- Start User Function (start)
  - Causes a transition from Idle mode to Active mode, using an internal event, Start_Ev

Required Functions & Events (cont.)

- Stop User Function (stop)
  - Causes a transition from Active mode or Pause mode to Idle mode
  - On the way from Active mode or Pause mode to Idle mode, the MMI module is responsible for closing the active layout

Required Functions & Events (cont.)

- Stopped Event (Stopped_Ev)
  - The result of calling the stop user function
  - Stopped_Ev should be triggered only when there is no danger that the module would write to the display

Required Functions & Events (cont.)

- Pause User Function (pause)
  - Initiates the Pausing process
  - Called by the main application when there is a high priority popup message that should be displayed
  - Can also be called by an HMI module for an active WMI module

Required Functions & Events (cont.)

- Paused Event (Paused_Ev)
  - The result of calling the pause user function
  - This event should be triggered only when the module is paused

Required Functions & Events (cont.)

- Resume User Function (resume)
  - Resumes the module after it was paused
  - The module should restore the last state that was active before it was paused

Required Modes for Widgets

- Basic mode tree for each widget component contains these modes:
  - **Idle:** component is awaiting activation
  - **Active:** contains the entire functionality of the active component

Required Functions & Events

- Initialization User Function (init)
  - Initializes the widget's holders
  - Called by the Layout Manager, SRV_LAYMAN.UDO
- Start User Function (start)
  - Causes a transition from Idle mode to Active mode, using an internal event, Start_Ev

Required Functions & Events (cont.)

- Stop User Function (stop)
  - Causes a transition from Active mode to Idle mode
  - Widget's responsibility is to clear its allocated area on the layout

Required Functions & Events (cont.)

- Stopped Event
  - The result of calling the stop user function
  - Should be triggered only when there is no danger that the module would write to the display

Required Functions & Events (cont.)

- Set Rectangle User Function (setRectangle_x:y:width:height:)
  - Sets the widget rectangle's position and size

Notes:

e-SIM MMI
Reference Design 1.x

Customization Tools

Introduction

Three external tools:

- The Layout Management Solution
- The Menu and Text Generator
- The Multimedia Resource Manager

These tools enable changes to aspects of a project without changing the RapidPLUS logic

Setting the eSim MMI

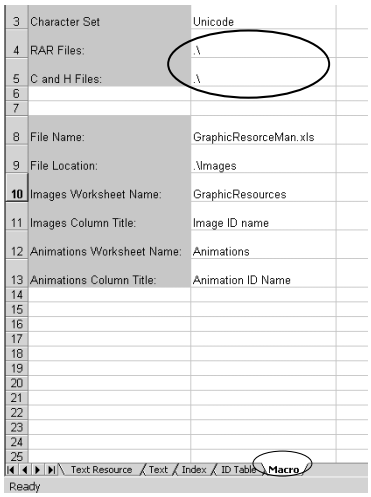
- Copy the LangAPI.DLL to c\Windows\System32
- Copy the 5 fonts from the fonts folder to controlPanel\Fonts
- Open the application from the RapidApp_Simulation folder

Setting the eSim MMI

- Open the TextResource.xls file
- Set the tools\security to Medum and reopen the file

Setting the eSim MMI

- Set the path as:



The Layout Management Solution

A two-part solution for defining and arranging the visual elements that appear on a product's screen

Layout Management Solution

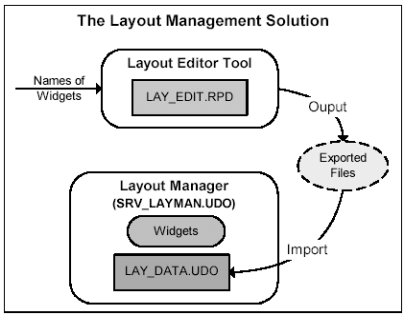
- A **layout** is a group of visual elements that have defined positions and sizes on a product's screen
- Layouts are designed in an external RapidPLUS application— the **Layout Editor** tool
- The Layout Editor generates data files that are used by the RapidPLUS project

Layout Management Solution (cont.)

- The layouts are managed during runtime using the service component, SRV_LAYMAN.UDO, which is referred to as the **Layout Manager**

Layout Management Solution (cont.)

- The architecture



- Exported files, located in the Layout_Editor\export folder:

lo_vt_data.rar
lo_vt_entry.rar
lo_vt_enum.h

The Menu and Text Generator

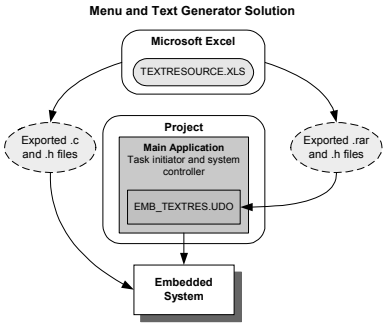
Manages multilingual menus and text resources

Menu and Text Generator

- Based on an Excel workbook that contains all of a project's strings, translations, and menu hierarchy information
- Contains macros for generating data files that are used by the RapidPLUS project and the embedded system

Menu and Text Generator (cont.)

- The architecture

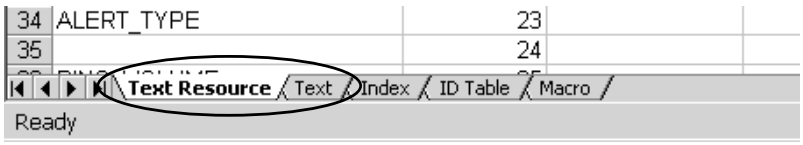


- Exported files for the RapidPLUS project:
  - CodePage.rar
  - Menu_Index.rar
  - Menu_Unicode.rar
  - LoadTo_ConstantSet.h
- Exported files for the embedded system:
  - cLng_Chinese_Simplified.c
  - cLng_English.c
  - cLng_French.c
  - cLng_German.c
  - LoadTo_ConstantSet.h
  - xl_IndexArray.c
  - xl_menu.h

Menu and Text Generator (cont.)

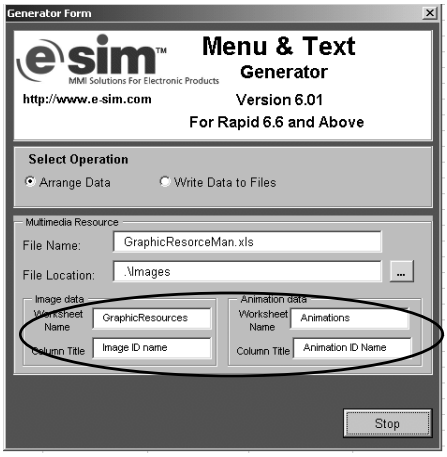
The Excel file:

- Change only the Text Resource and Text worksheets (tabs)



Menu and Text Generator (cont.)

- Run the macro at the end



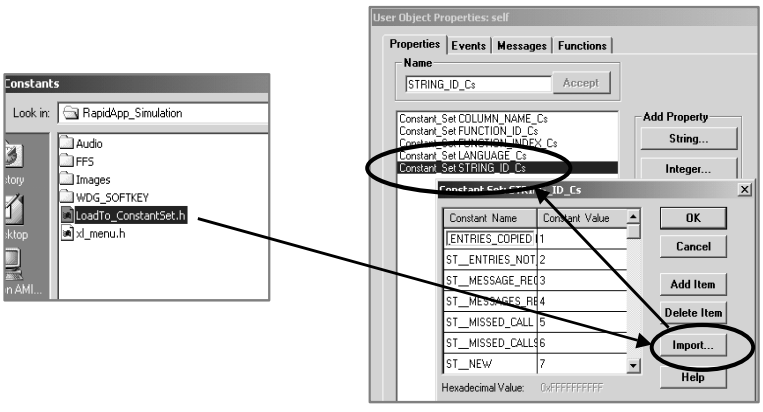
For more details, refer to the Menu and Text Generator Solution.pdf

Menu and Text Generator (cont.)

- The output file LoadTo_ConstantSet.h contains:
  - String IDs—for string usage
  - Function Index—for the top list item at each submenu
  - Function IDs—function names to be executed when activating a specific feature

Menu and Text Generator (cont.)

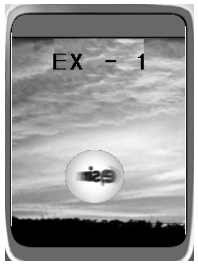
- Import constants into EMB_TXTRES.UDO:



esim

Exercise 1: Text Generator

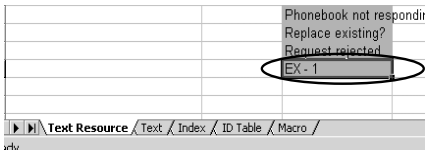
- Change the welcome message to EX – 1
- Change the positions of the message and animations



esim

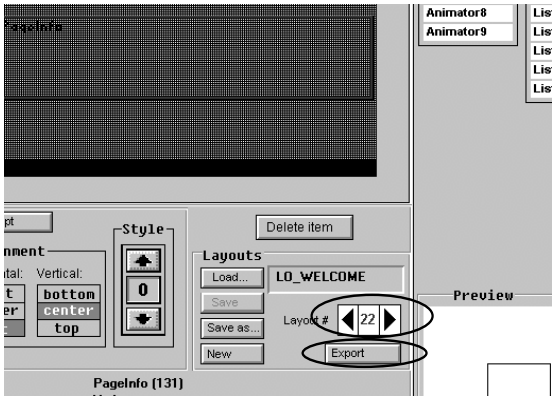
The Solution: Step A

- Add a new text string to the Excel workbook, TextResource.xls
- Run the macro



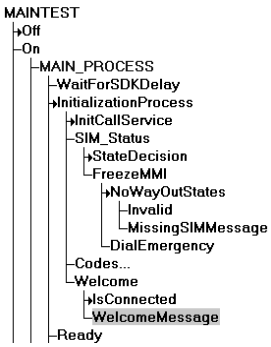
The Solution: Step B

- Change Layout #22
- Export the layout



The Solution: Step C

- Import constant set into EMB_TXTRES.UDO
- Edit the code inside MAINTEST.RPD



The Multimedia Resource Manager

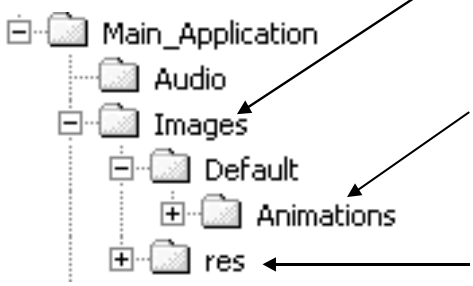
Manages bitmaps and animations

Multimedia Resource Manager

- Based on an Excel template that contains macros for generating IDs for all of the bitmaps
- These generated IDs are used by several RapidPLUS components
- All of the project bitmaps are organized in a separate folder

Multimedia Resource Manager (cont.)

- Folder hierarchy



All the bitmaps that are used in a project must be all together in a subfolder

Animations must be in a subfolder that holds only animation files

Used by the Excel template as the destination folder

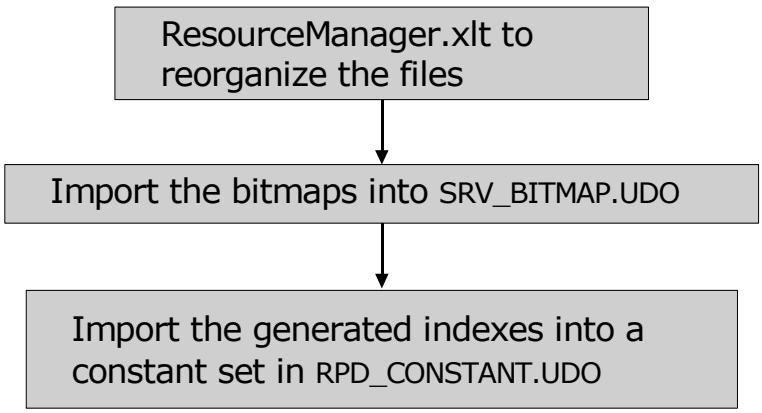
Multimedia Resource Manager (cont.)

- Bitmap file names
  - File names must not contain blank spaces
  - Each bitmap must have a unique name, even though the original bitmap files are located in different folders
  - The file names will be used in the constant set in RPD_CONSTANTS.UDO

Multimedia Resource Manager (cont.)

- **SRV_BITMAP.UDO**
  - Contains an array of objects that holds the bitmaps, which are arranged by index number (as defined in the Excel worksheet)
  - Array elements are imported from the bitmaps in the destination folder
- **RPD_CONSTANTS.UDO**
  - Has a constant set that contains a constant item for each bitmap. The constant set items are imported from the Excel-generated .h file

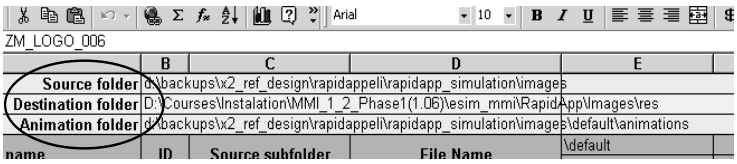
Multimedia Resource Manager (cont.)



Multimedia Resource Manager (cont.)

ResourceManager.xlt to reorganize the files

Make sure the paths are correct



Files generated to the res folder

Multimedia Resource Manager (cont.)

Import

- Copies all of the bitmaps to the Destination folder (res)
- Imports all of the bitmaps into Excel and assigns an ID name and number to each bitmap
- Creates various files, including an .h file that can be imported into RapidPLUS
- Organizes the animation folders into a separate worksheet

Multimedia Resource Manager (cont.)

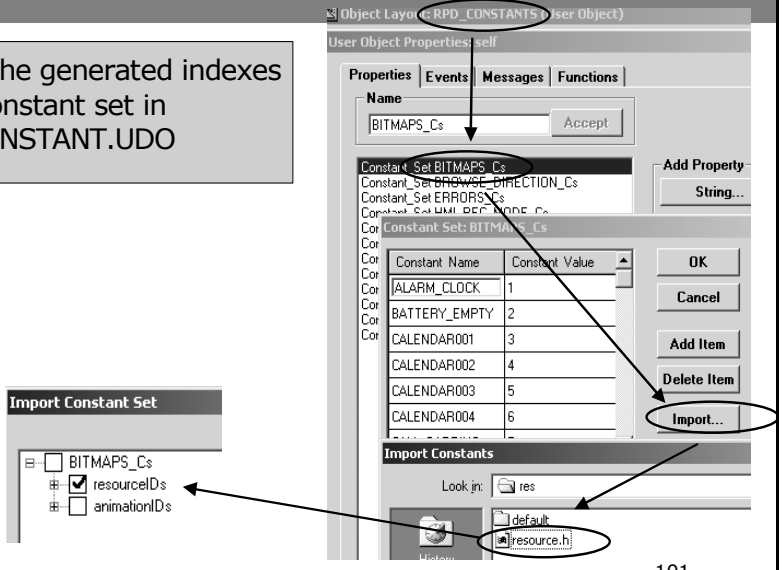
Import the bitmaps into SRV_BITMAP.UDO

If you add a new bitmap to the Source folder:

1. Use the File|Import Bitmaps command to import all of the images from the res\default folder
2. In the "Import Bitmaps from" dialog box, select the Link to File option

Multimedia Resource Manager (cont.)

Import the generated indexes into a constant set in RPD_CONSTANT.UDO



Notes:

e-SIM MMI
Reference Design 1.x
Synchronization of Components

Main Application as Task Manager

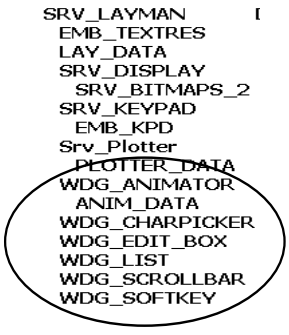
- The main application manages the entire project by activating and deactivating the MMI modules
- Only one MMI module is active at a time
- Only the active MMI module displays on the screen

Main Application as Task Manager (cont.)

- To activate one module at a time, the "active" mode (named MainMMI) is organized into exclusive modes, one for each MMI module
- When the active MMI module's task has been completed it terminates itself and notifies the main application (by triggering the Stopped_Ev exported event)
- The main application receives Stopped_Ev and returns to its Idle mode

Working with Widgets

- Widgets reside under SRV_LAYMAN.UDO, the Layout Manager service component
- Widgets are controlled by the Layout Manager and by MMI modules that require the widgets' functionalities



Working with Widgets (cont.)

When an MMI modules requires a widget:

1. When the init_layout function is called, each widget that participates in the layout is initialized by the Layout Manager
2. Each widget is specifically initialized by the MMI module with the relevant data that is required for operation
3. Each widget is started by the Layout Manager immediately after the Layout Manager's start function is called by the MMI module

Working with Widgets (cont.)

4. After a layout is started, the MMI module directly manipulates the widget according to the widget's specific interface
5. When the layout is no longer required, the MMI module calls the Layout Manager's stop function, which initiates each widget's Stopping process

Pause and Resume Mechanism

- Pause and resume mechanisms allow high priority popup messages to be displayed
- When an active MMI module is interrupted for a popup message, the main application:
  1. Creates a list of MMI modules to be paused. Only modules that are active—and not paused—at the moment the process begins would be entered on the list

Pause and Resume Mechanism (cont.)

  2. Creates a list of objects to be resumed
  3. Pauses relevant components
  4. Displays the popup message
  5. Resumes the relevant modules
  6. Restores the state to where it was before the pause process began

esim

MMI Modules - Notes

- The MMI modules should have no graphic action in its pause mode
- When the module is resumed, it may use a deep history transition to return to the previous state
- Initializing widget data should be performed on actions of the transition entering the mode and not on entry activities, in order to be able to re-enter the mode without re-initializing the data
- Avoid using a junction mode (may cause a conflict with the deep history transition)

Notes:

e-SIM MMI
Reference Design 1.x

The Draw Method

Objectives

- How to draw
- Add a new layout

How to Draw in the MMI

How to Draw in the MMI (cont.)

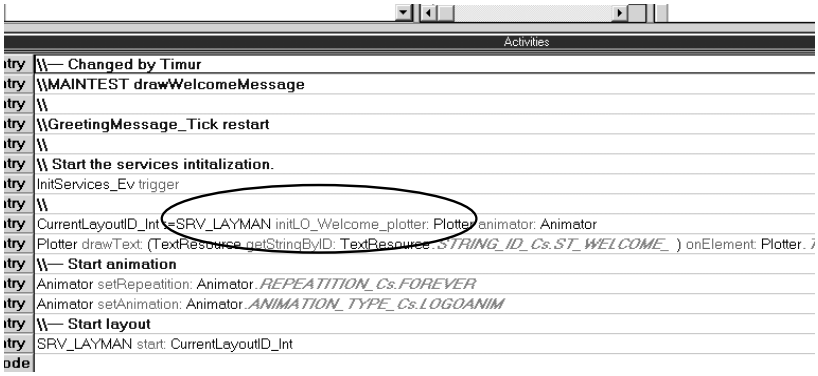
- An HMI module calls the plotter service using the function:

  Plotter drawText: onElement:

- To use the plotter you need to specify a layout
- The plotter calls the Display

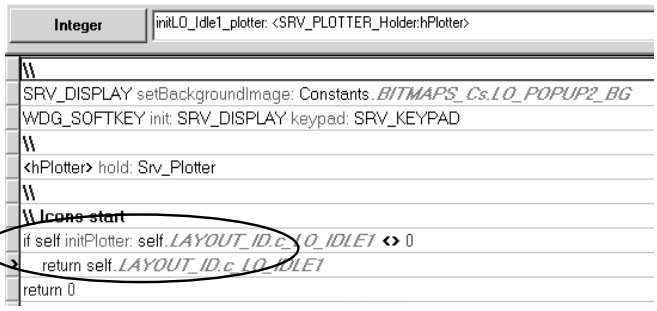
Order Of Execution: Draw

MAINTEST.RPD/HMI:



Order Of Execution: Draw (cont.)

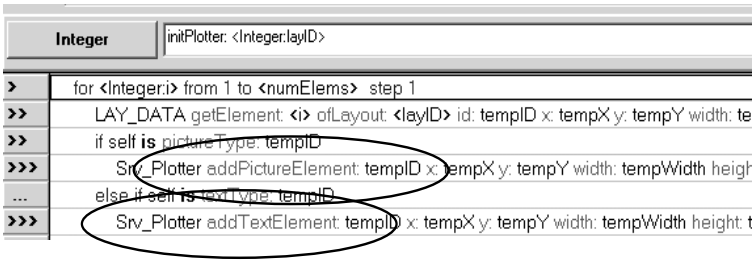
SRV_LAYMAN.UDO:



Order Of Execution: Draw (cont.)

SRV_LAYMAN.UDO:

Update the plotter on which type of elements it will hold



Order Of Execution: Draw (cont.)

MAINTEST.RPD/HMI:

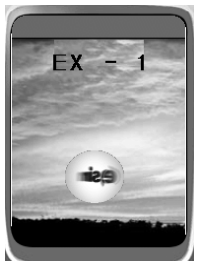
Order Of Execution: Draw (cont.)

PLOTTER.UDO:

```
Integer | drawText:<String:text>onElement:<Integer:elementID>
\\ Draw the text for the text element specified by elementID.Return 1 if succeeded. Return
\\
declare <Integer:cnt>
    <cnt> := PLOTTER_DATA.All.Elements.textCount
    for <Integer:i> from 1 to <cnt> step 1
        if PLOTTER_DATA.All.Elements.textElement[ <i> ].id = <elementID>
            Display clearAreaAtx: PLOTTER_DATA.All.Elements.textElement[ <i> ].X y: PLOTTER_
            Display setStyle: PLOTTER_DATA.All.Elements.textElement[ <i> ].style
            Display drawText: <text> tofitRect_: PLOTTER_DATA.All.Elements.textElement[ <i> ].X
            PLOTTER_DATA.All.Elements.textElement[ <i> ].text := <text>
            self updateDisplay
            return 1
return 0
```

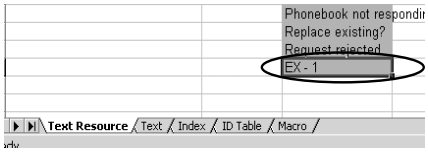

Exercise 1: Draw Method

- Change the welcome message to EX – 1
- Change the positions of the message and animations



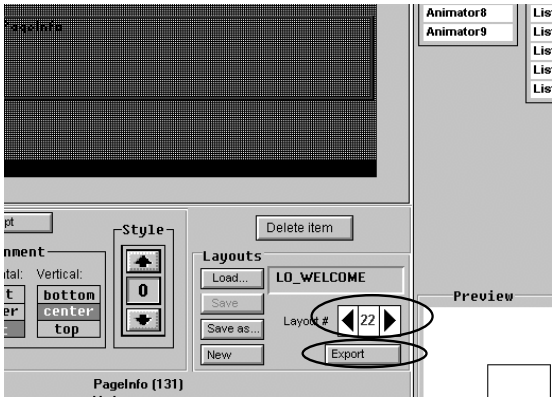
The Solution: Step A

- Add a new text string to the Excel workbook, TextResource.xls
- Run the macro



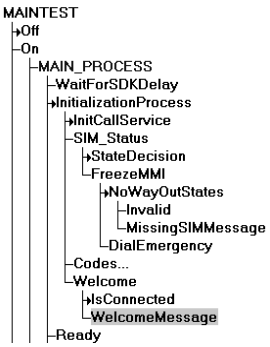
The Solution: Step B

- Change Layout #22
- Export the layout



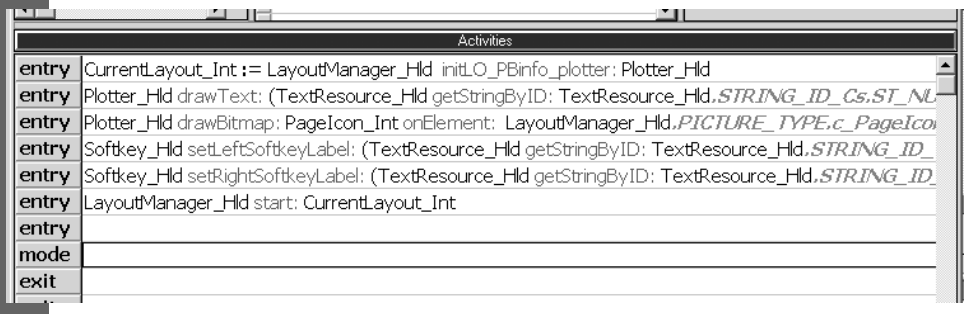
The Solution: Step C

- Import constant set into EMB_TXTRES.UDO
- Edit the code inside MAINTEST.RPD

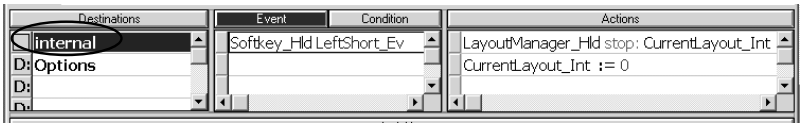


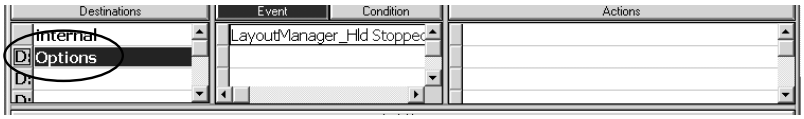
Draw - summary

- Example for normal page:



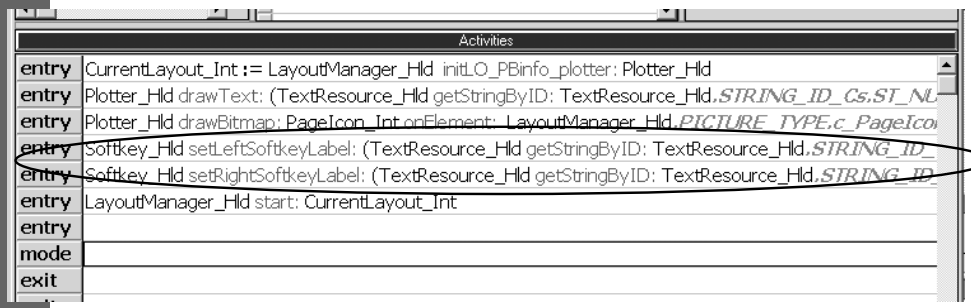
•Close the Layout:





Using the Soft keys:

•Use inside a layout



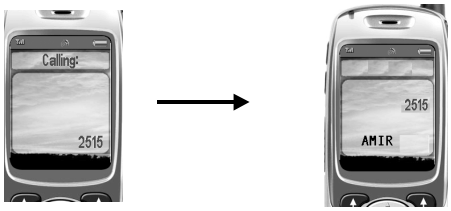
Using the Soft keys:

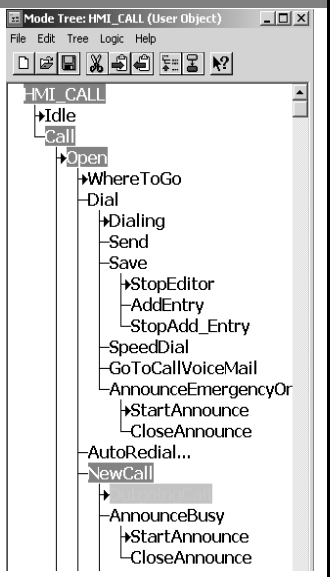
- Use inside a Menu

K	L	M	N	O	
		String	String	Integer	String
el9	Level1[illegible]	LeftSK	RightSK	Layout	Image ID na
		OK	Back	23	
		OK	Back	23	
		OK	Back	23	
		Select	Back	23	
		Select	Back	23	CALL_BARRI
		Select	Back	23	
		OK	Back	23	

Exercise 1A: Draw Method

- Change the OutgoingCall screen Title to: your name
- Change the positions of the Title and TextBox



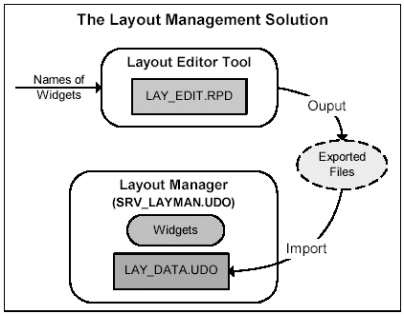


Exercise 2: Add a New Layout

- The layout will contain a TextBox and a bitmap
- Use the layout for the welcome message screen

Exercise 2: Add a New Layout (cont.)

- The architecture



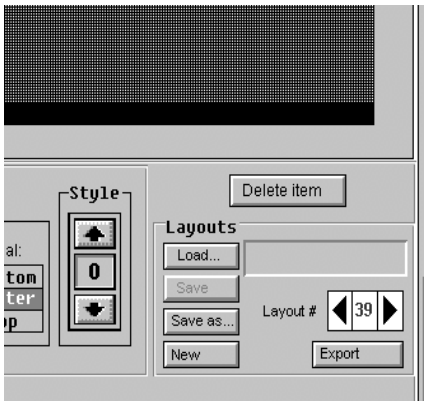
- The exported files

lo_vt_data.rar
lo_vt_entry.rar
lo_vt_enum.h

(located in the Layout_Editor\export folder)

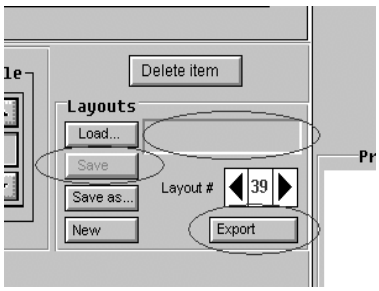
The Solution: Step A

- Create new layout in the Layout Editor
- Place the TextBox and bitmap elements



The Solution: Step B

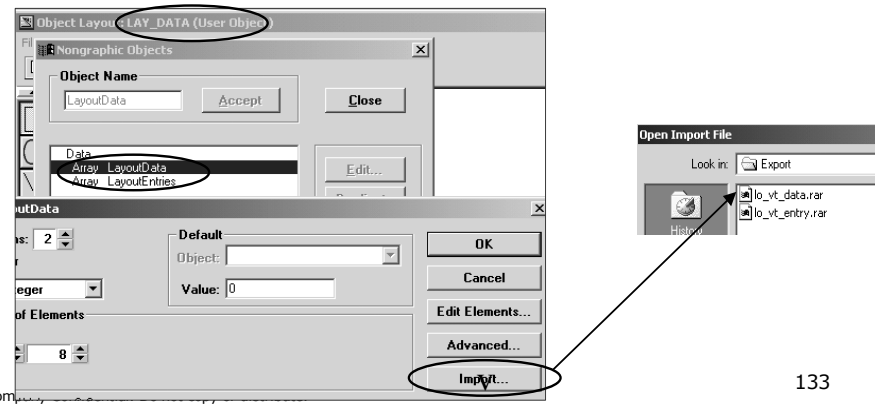
- Name the layout
- Save the project, then export it



(The exported files are updated)

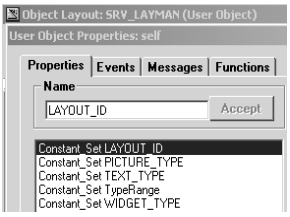
The Solution: Step C

- **LAY_DATA.UDO**:
  - Update the LayoutData array by importing lo_vt_data.rar
  - Update the LayoutEntries array by importing lo_vt_entry.rar



The Solution: Step D

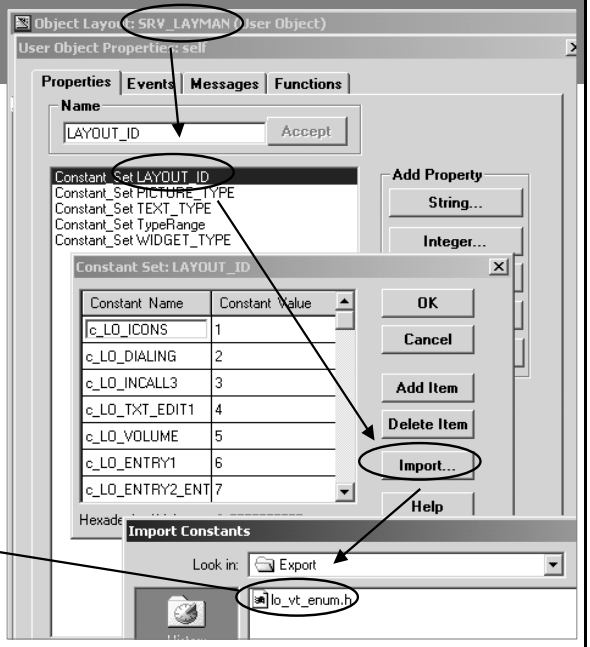
- **SRV_LAYMAN.UDO**
  - The constant set properties: LAYOUT_ID, PICTURE_TYPE, TEXT_TYPE, and WIDGET_TYPE have corresponding enumerated sets in the file lo_vt_enum.h.
  - If you have only added a new layout, only LAYOUT_ID must be updated

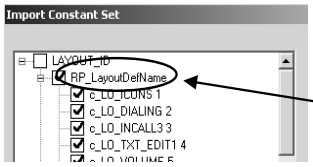


esim

Step D (cont.)

- Import lo_vt_enum.h
- Select RP_LayoutDefName

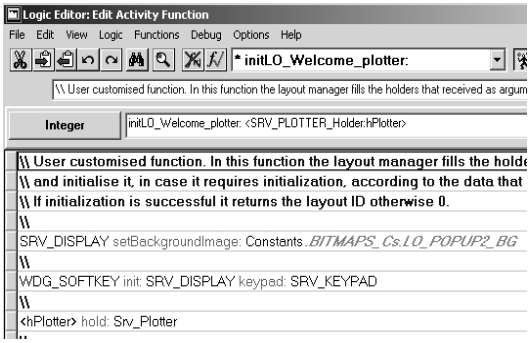




esim

The Solution: Step E

- **SRV_LAYMAN.UDO**
  - Add an init function for the layout

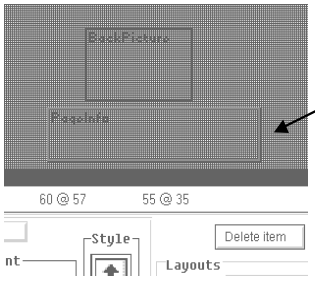


- **MAINTEST.RPD**

The Solution: Step F

- Use the drawText: function with the correct layout element
- MAINTEST.RPD:

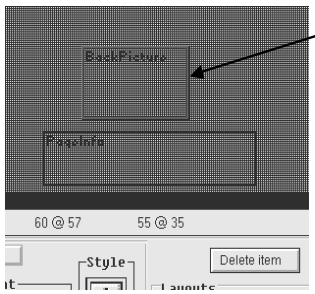
```
ntry  CurrentLayoutID_Int :=SRV_LAYMAN initLO_Welcome_plotter: Plotter animator: Animator
ntry
ntry  Plotter drawText: (TextResource getStringByID: TextResource.STRING_ID_Cs.ST_WELCOME_ ) onElement: Plotter. TEXT_TYPE.c_PageInfo
ntry
ntry  Plotter drawBitmap: Constants.BITMAPS_Cs.ALARM_CLOCK onElement: Plotter.PICTURE_TYPE.c_BackPicture
```



The Solution: Step G

- Use the drawBitmap: function with the correct layout element
- MAINTEST.RPD:

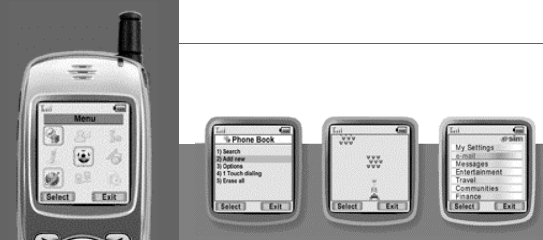
```
ntry  CurrentLayoutID_Int :=SRV_LAYMAN initLO_Welcome_plotter: Plotter animator: Animator
ntry
ntry  Plotter drawText: (TextResource getStringByID: TextResource.STRING_ID_Cs.ST_WELCOME_ ) onElement: Plotter. TEXT_TYPE.c_PageInfo
ntry
ntry  Plotter drawBitmap: Constants.BITMAPS_Cs.ALARM_CLOCK onElement: Plotter.PICTURE_TYPE.c_BackPicture
```



Summary

You have learned to modify existing layouts and add new layouts!

Notes:



e-SIM MMI
Reference Design 1.x
Menu Method

Objectives

- Learn how to change the menu
- Learn how to add a menu leaf
- Learn how to add a new Options Menu

Types of Menus

- The animation icons menu- Main Menu
- The Submenu
- Options menu- specific for each MMI

MMI Modules

HMI_MENU

- Handles the main menu and the submenu
- After selecting a submenu the selected HMI will start its own menu

WMI_MENU

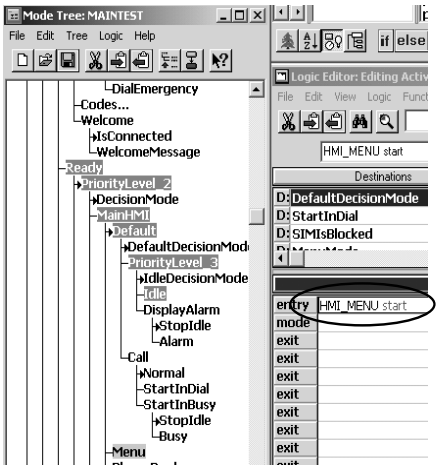
- Handles navigation to all menu items
- Once activated, menu navigation becomes automatic

The Menu Mechanism

Main Menu

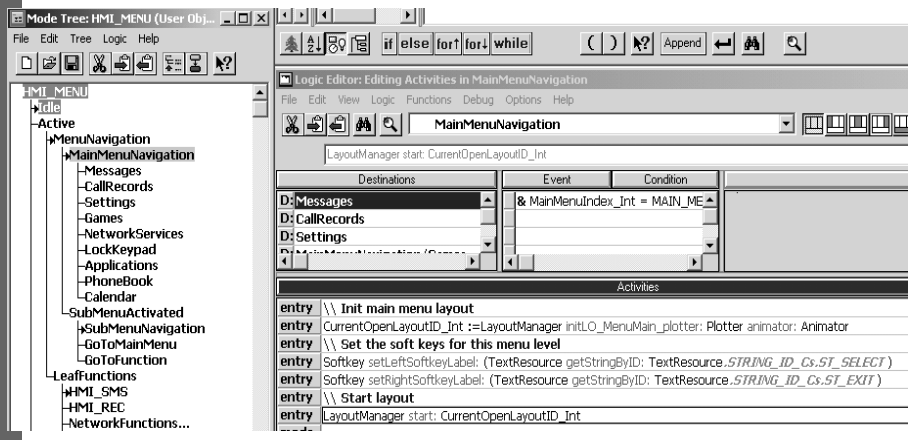
The Menu Mechanism - Main Menu

- First call from MAINTEST.RPD:



The Menu Mechanism - Main Menu (cont.)

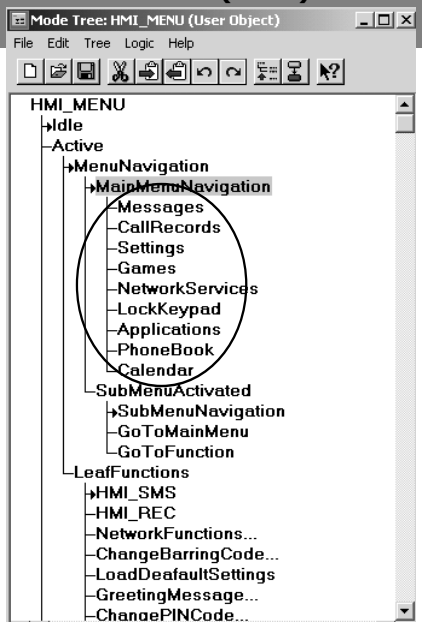
- HMI_MENU:



The Menu Mechanism - Main Menu (cont.)

HMI_MENU:

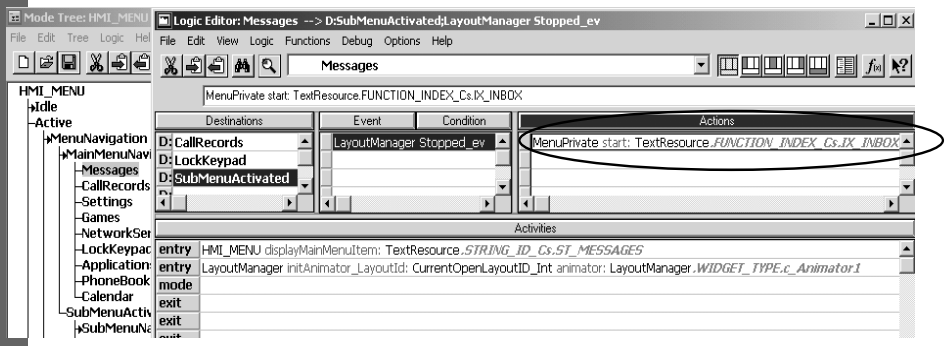
- A main menu item is implemented as a mode in HMI_MENU.UDO
- Navigation of the main menu is done using WMI_MENU_PRIVATE
- Implementation of menu functionality is done in HMI_MENU.UDO



The Menu Mechanism

Submenu

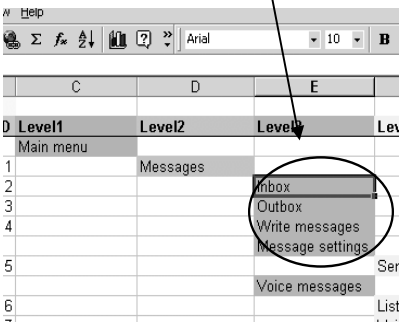
The Menu Mechanism - SubMenu



- Transition to SubMenuActivated mode allows navigation in a submenu by start MENU_PRIVATE (WMI_MENU.UDO)

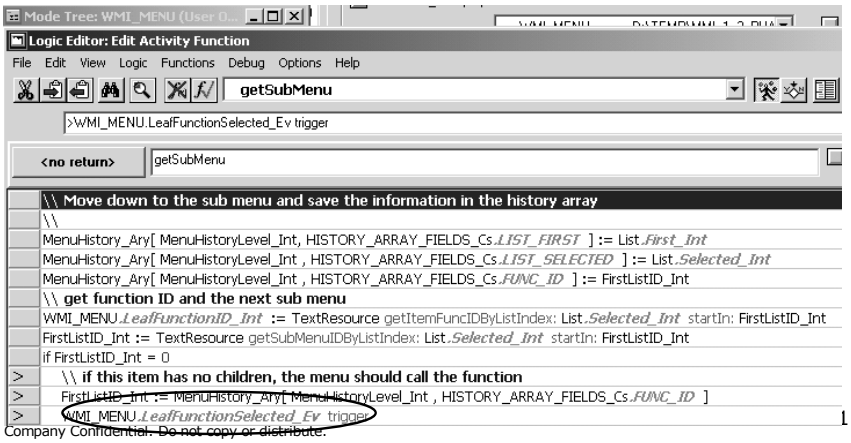
The Menu Mechanism - SubMenu (cont.)

- WMI_MENU starts the new menu from the first function index item



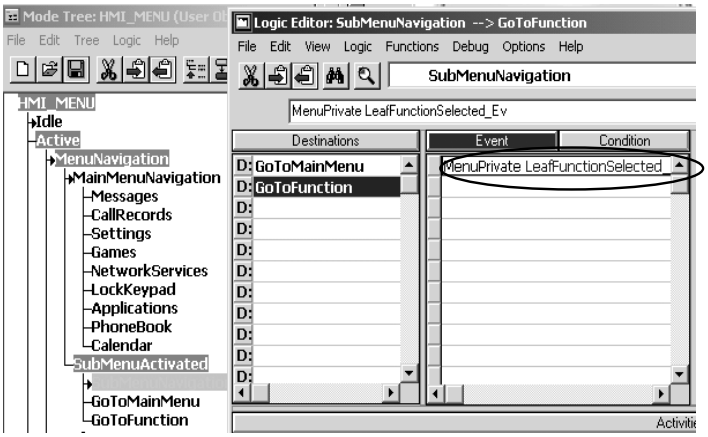
The Menu Mechanism - SubMenu (cont.)

- When WMI_MENU navigation reaches Menu Leaf it will turn the HMI_MENU to the function mode



esim

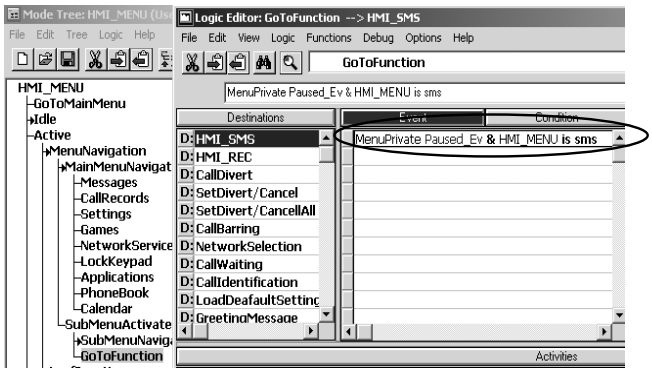
The Menu Mechanism - SubMenu (cont.)



esim

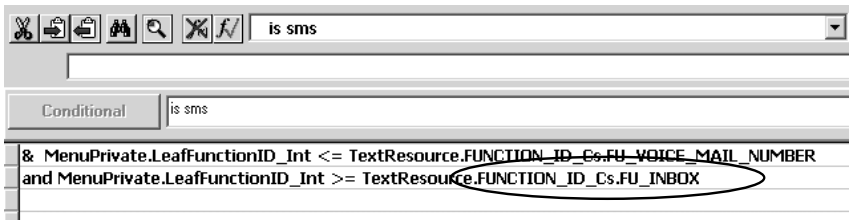
The Menu Mechanism - SubMenu (cont.)

- Combination of WMI_MENU_PRIVATE's paused event and a condition on its leaf function ID property



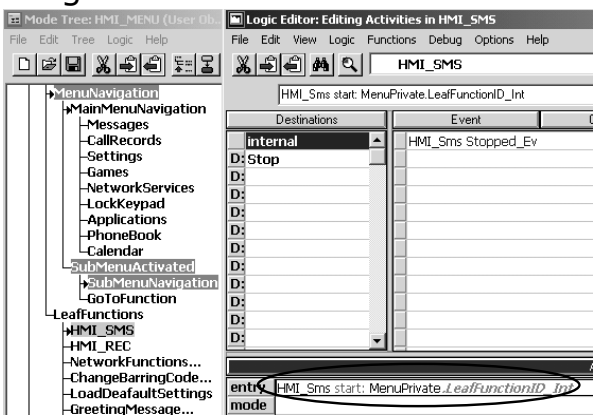
The Menu Mechanism - SubMenu (cont.)

- Leaf function ID should equal the function ID that you assigned to the implemented function from FUNCTION_ID_Cs (located in EMB_TEXTRES.UDO)



The Menu Mechanism - SubMenu (cont.)

- Starting the HMI



- From that point the HMI will take control

More Mechanisms

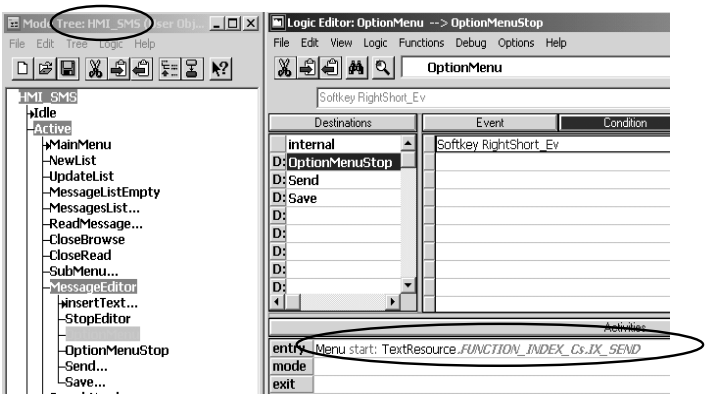
Add Options Menu

Change Menu Item

Add Menu Item

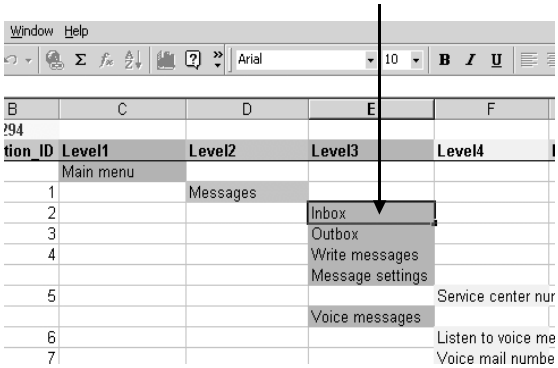
Add Options Menu

- Each HMI will start its own options menu



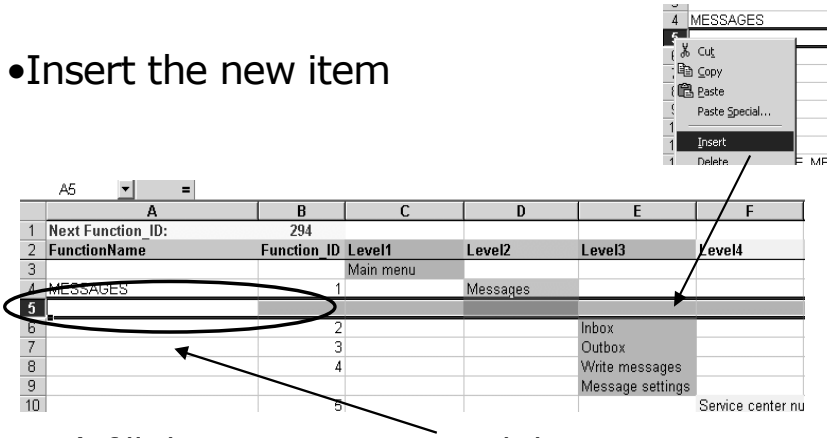
Change Menu Item

- Change the string in TextResource.xls



Add Menu Item

- Insert the new item

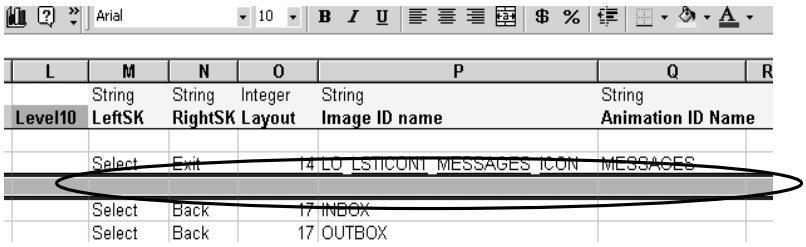


- Don't fill the FunctionName and the Function_ID columns

Add Menu Item (cont.)

- Fill the following fields:

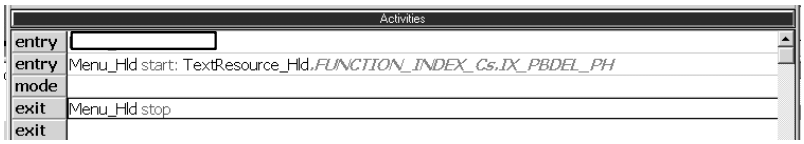
LeftSK , RightSK, Layout,ImageID,Animation ID



L	M	N	O	P	Q	R
	String	String	Integer	String	String	
Level10	LeftSK	RightSK	Layout	Image ID name	Animation ID Name	
	Select	Exit	14	LO_LSTICONT_MESSAGES_ICON	MESSAGES	
	Select	Back	17	INBOX		
	Select	Back	17	OUTBOX		

Menu Summary 1

- Example for menu managing:

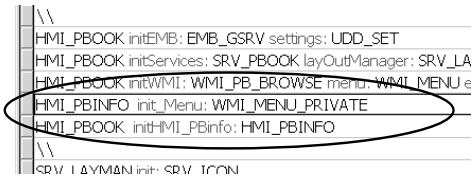


WMI_MENU_PRIVATE

Optional

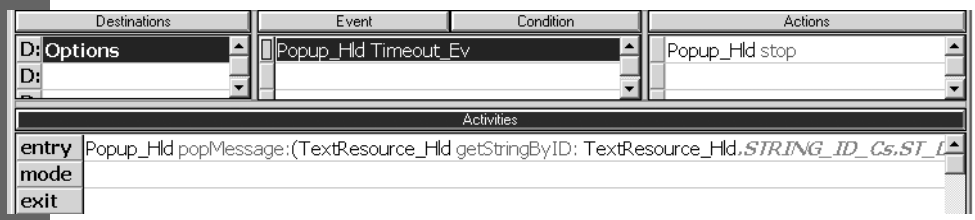
- you can use another instance of wmi_menu, by init your HMI with the WMI_MENU_PRIVATE (it is a copy of the wmi_menu

MainTest.RPD :



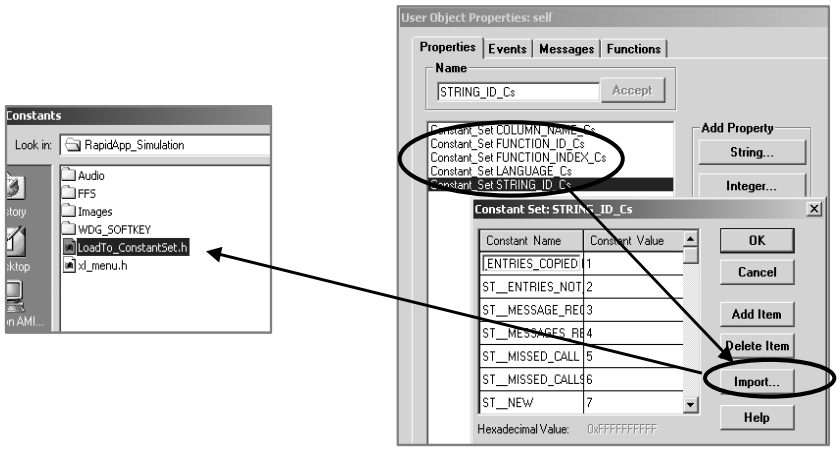
Step 1 : New HMI

- Example for Popup



Add Menu Item (cont.)

- Import into RapidPLUS (EMB_TXTRES.UDO)



Menu 1 Exercise

Details:

- Add Menu Item
- Create a new item in the Submenu
- Add the item "MySMS" to the Messages menu
- The final result will be:

Menu 2 Exercise

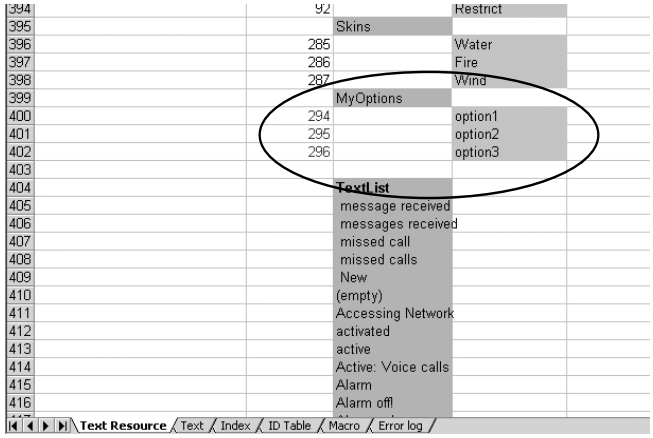
Details:

- Add Options Menu
- The menu will be located on the MemoryStatus screen

Menu 2 Exercise (cont.)

- Add a new Options menu
- Only the menu navigation will function
- Back will go to **Idle screen**

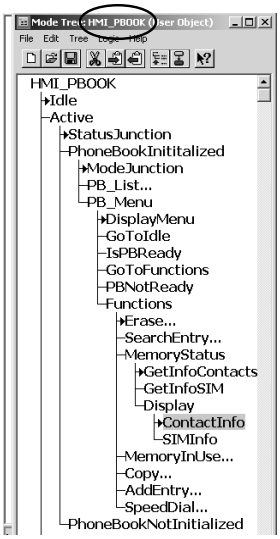
•Create new options menu



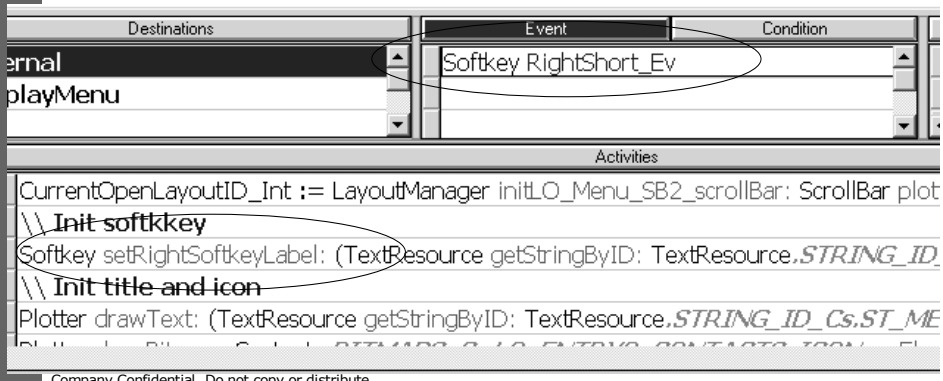
394		92	Restrict
395		Skins	
396		285	Water
397		286	Fire
398		287	Wind
399		MyOptions	
400		294	option1
401		295	option2
402		296	option3
403			
404		TextList	
405		message received	
406		messages received	
407		missed call	
408		missed calls	
409		New	
410		(empty)	
411		Accessing Network	
412		activated	
413		active	
414		Active: Voice calls	
415		Alarm	
416		Alarm offl	

Text Resource / Text / Index / ID Table / Macro / Error log /

•Add new mode for the Option menu

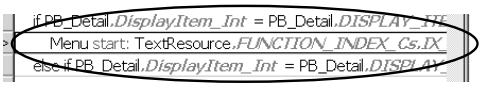


- Use the Softkey



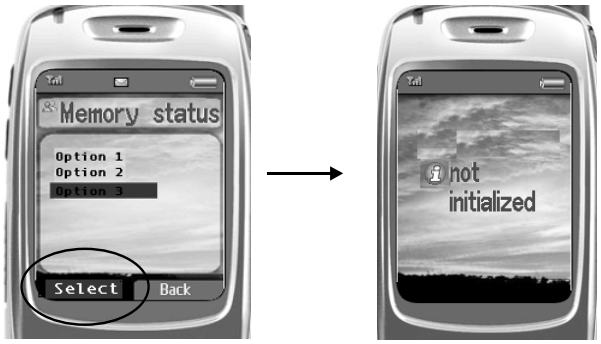
- Stop the curentLayout when press Options and then start the options menu (Menu start ,

calling the functionIndex)

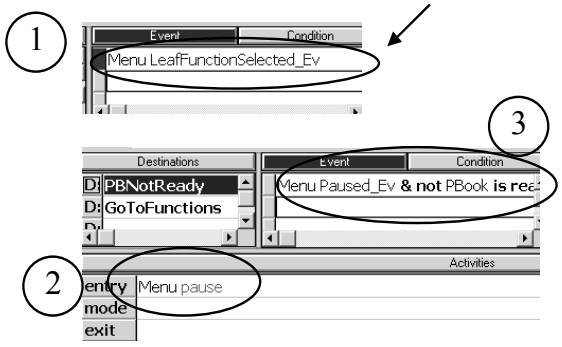


Popup Exercise

- Choosing Option3 will display 2 sec Message
- Use Popup timeout_Evt for the timing

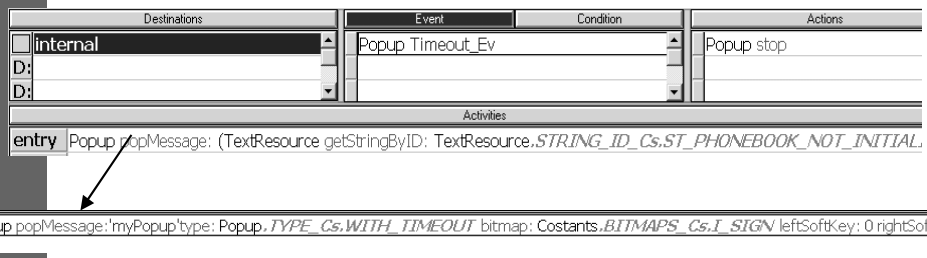


(On each LeftSoftKey press the menu generate):



Pop Up Message

- Using the WMI_Popup.UDO
- Timeout_Ev
- Stop function , Then call menuResume function



Step 1 : Final Ex. New HMI

Define the new HMI and the screens only

Don't display the memory data

See FinalEx.Doc

Step 2 : Manage the Memory

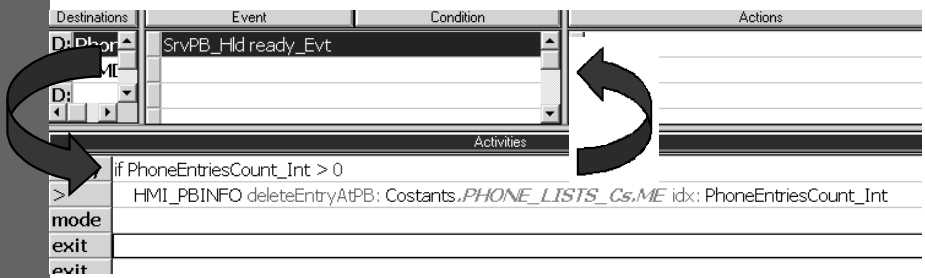
- **How to Read the memory:**

1. Create holder to SRV_PBOOK.UDO
2. Call the usedOfIn function
3. Wait for ready_Evt
4. Read the memory from Results.usedOf.used

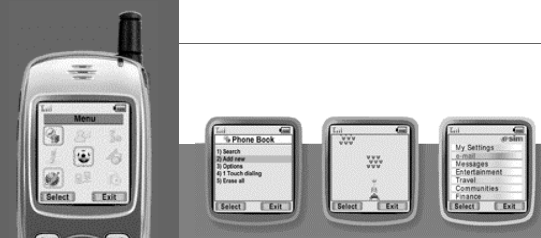
Step 2 : Manage the Memory

- **How to Delete the memory:**

1. Use the SRV_PB
2. Call deleteEntryFrom: <pb> at: <idx>
3. Wait for ready_Evt and then call it again –Loop



Notes:



e-SIM MMI
Reference Design 1.x

Final Exercise

Final Exercise

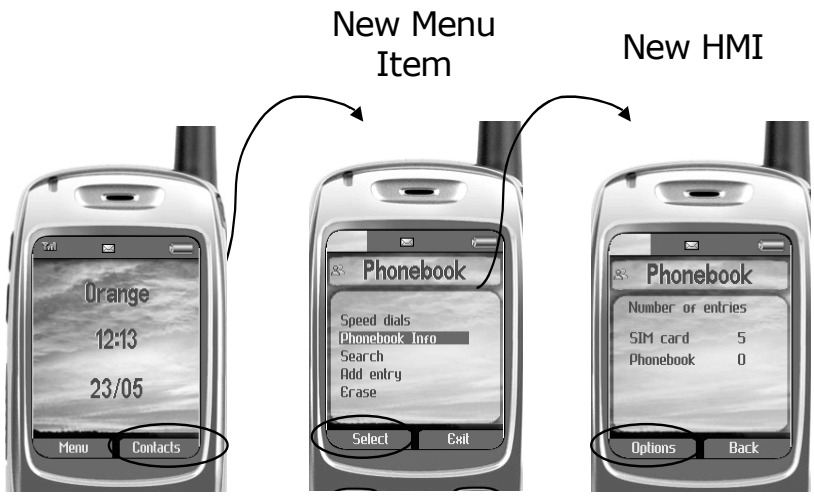
Goal:

- Add a new MMI including Options menu

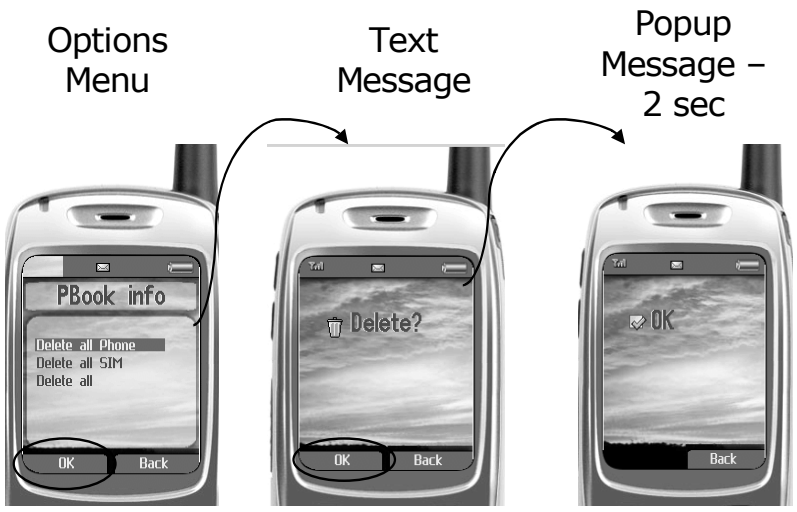
Details:

- Add a new layout
- Add a new menu leaf to textResource
- Add a new options menu to textResource
- Start the new HMI from HmiMenu
- Start the options menu from the new HMI

Final Exercise – Screen Shots



Final Exercise – Screen Shots (cont.)



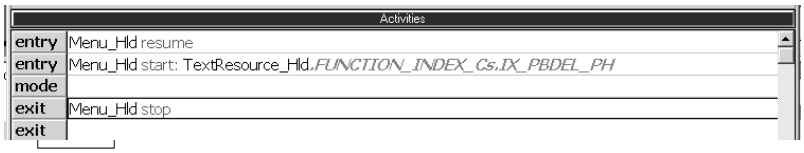
Step 1 : New HMI

Define the new HMI and the screens only

Don’t display the memory data

Step 1 : New HMI

- Example for menu managing:



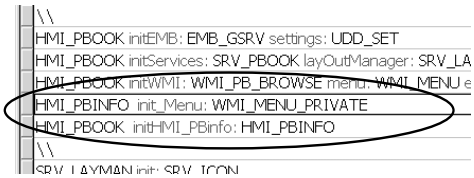
- We call the **resume** because the menu is in pause from the PBOOK.

WMI_MENU_PRIVATE

Optional

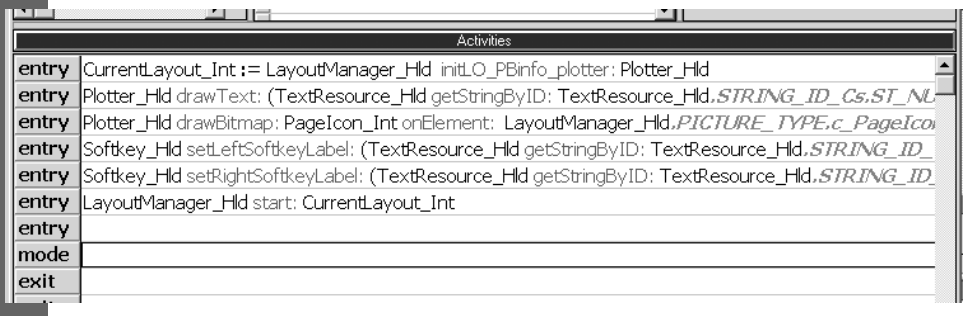
- you can use another instance of wmi_menu
  - Init the PBInfo menu with another menu because the wmi_Menu is still in pause from the Pbook

MainTest.RPD :



Step 1 : New HMI

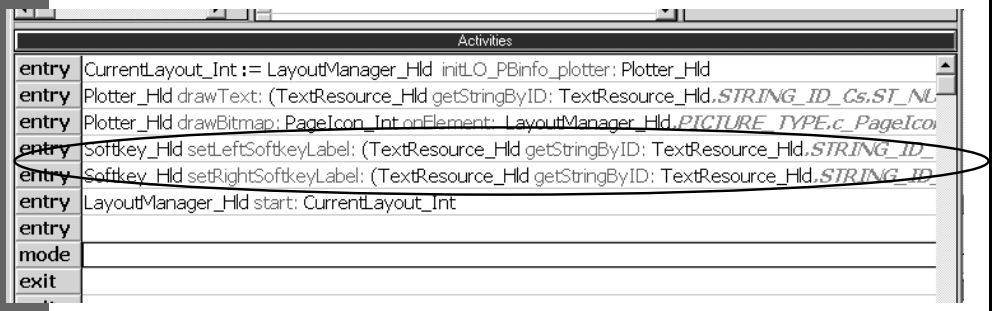
- Example for normal page:c



esim

Using the Soft keys:

- Use inside a layout



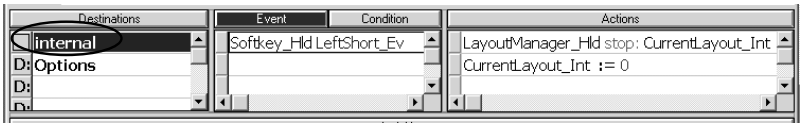
esim

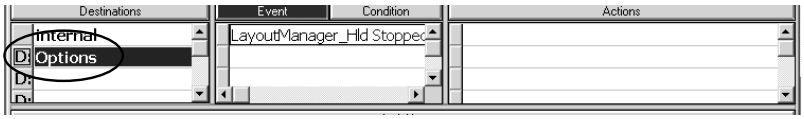
Using the Soft keys:

- Use inside a Menu

K	L	M	N	O	
		String	String	Integer	String
el9	Level18	LeftSK	RightSK	Layout	Image ID na
		OK	Back	23	
		OK	Back	23	
		OK	Back	23	
		Select	Back	23	
		Select	Back	23	CALL_BARRI
		Select	Back	23	
		OK	Back	23	

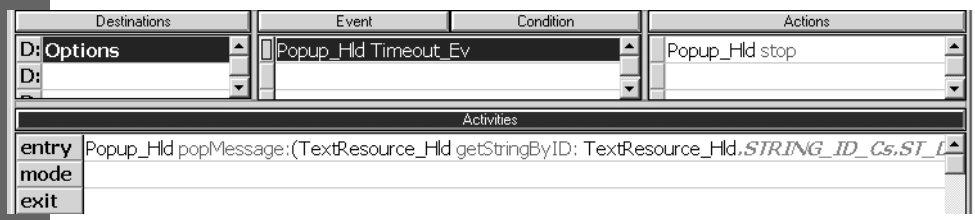
•Close the Layout:





Step 1 : New HMI

•Example for Popup



Step 2 : Manage the Memory

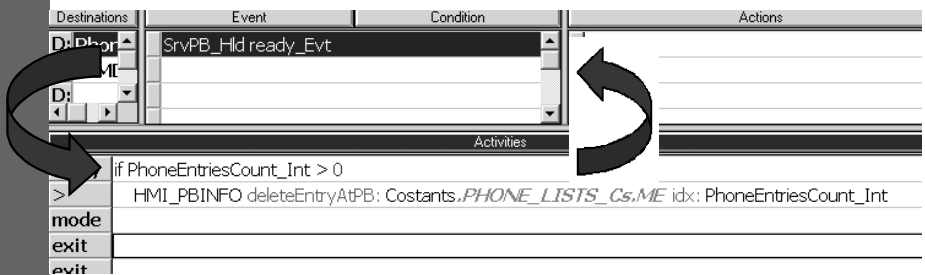
- **How to Read the memory:**

1. Create holder to SRV_PBOOK.UDO
2. Call the usedOfIn function (will generate Evt)
3. Wait for ready_Evt
4. Read the memory from Results.usedOf.used

Step 2 : Manage the Memory

- **How to Delete the memory:**

1. Use the SRV_PB
2. Call deleteEntryFrom: <pb> at: <idx>
3. Wait for ready_Evt and then call it again –Loop



Notes:

Notes:

Notes: