Belly up to the UltraToolBar and Order from the Menu

Paul Guggenheim

Paul Guggenheim & Associates

About PGA

- Working in Progress since 1984 and training Progress programmers since 1986
- Designed seven comprehensive Progress courses covering all levels of expertise including - The Keys to OpenEdge®
- Author of the Sharp Menu System, a database driven, GUI pull-down menu system.
- **White Star** Software Strategic Partner
- **TailorPro** Consultant and Reseller
- **Tools4Progress** Partner
- Major consulting clients include Acument Global Technologies, Cloverdale Paints, Foxwoods Casino, Great Valley Technologies, Indiana Packers Corporation, Interlocal Pension Fund, International Financial Data Services, National Safety Council, Plymouth Tube, Stripco and Tyson Foods.
- Head of the Chicago Area Progress Users Group

Overview

- Migrating from traditional GUI to GUI for .NET
- Using .NET Forms with ABL Windows
- Embedding ABL Windows in .NET Forms
  - Elements of an embedded ABL Window
  - Handling Form and Window Input
  - Single Window Persistent Procedure Example
  - ADM2 Paging Example

Overview (continued)

- Introduce UltraToolBar and components
  - UltraToolbar
  - UltraToolbar Manager
  - Popup-Menu
  - Button Tool
- Integrate UltraToolBar into existing ABL Application
  - OpenEdge Dynamic Menu
  - UltraToolBar Dynamic Menu Integration Example
  - Calling .NET Forms from the UltraToolBar
  - Walking the Form Tree
  - Calling UserControls from the UltraToolBar

Migrating to GUI for .NET

- OpenEdge provides many options to combine Windows and Forms in a single application.
- As a result, there is a lot of flexibility in balancing the level of migration versus integration.

Migrating to GUI for .NET

- The following table is a comparison between a GUI Window in the ABL versus a .NET Form

ABL Window	.NET Form
Windows are widgets	Forms are classes
Weak-typed handle-based object at run-time	Strong-typed class hierarchy at compile-time
Interact and contain other widgets only	Interact, contain and extend other widget classes

ABL Integration Features

- A Window may be embedded inside of a Form, allowing the window's widgets to be displayed.
  - The Window has to be unrealized before it is embedded in a Form.
- Both the Window and the Form will use a single Wait-For statement.
  - e.g. wait-for Application:Run(MainForm).
  - Both can also use the Process Events statement.
- OpenEdge provides a single chain of objects to track both Windows and Forms (Form Chain).

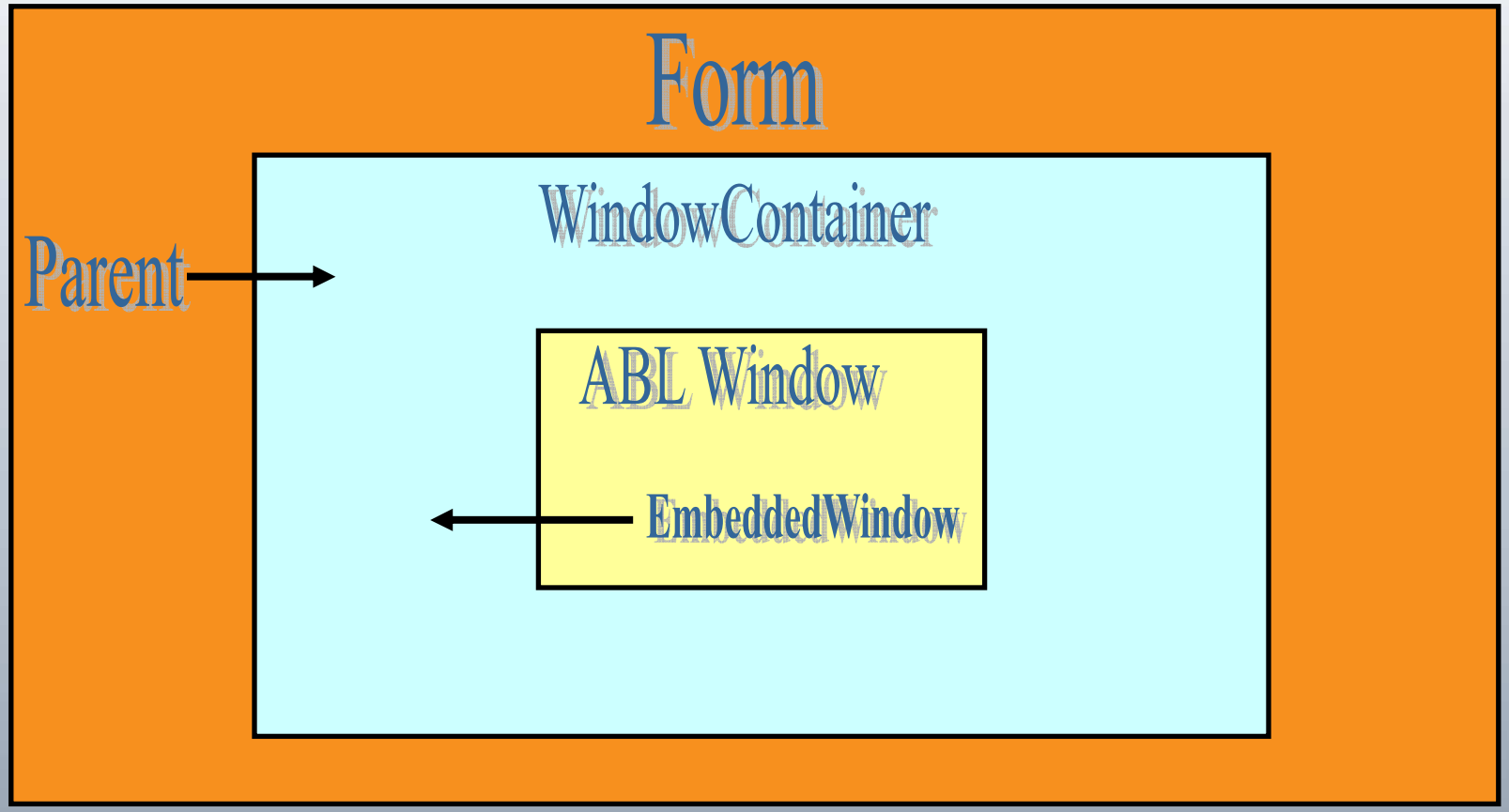
Embedded Window Elements

- All Frames and their corresponding child widgets that are part of the Window are included in the client area of the Form.
- The following items are NOT included:
  - Menu and submenu widgets
  - Message Area
  - Status Area
  - Title Area

WindowContainer Class

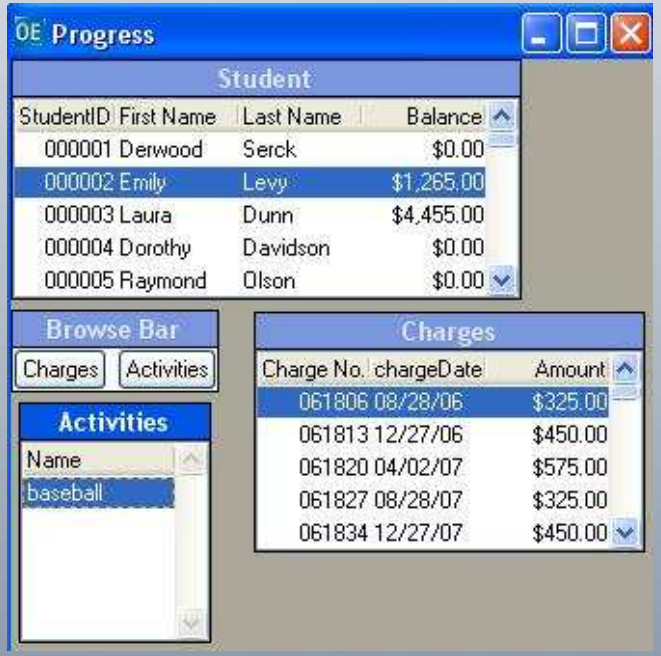
- To connect a Window to a Form, OpenEdge provides an intermediate object called the WindowContainer class.
- It contains two key properties.
  - EmbeddedWindow – Assigns the handle to the Window that is to be embedded.
  - Parent – Assigns the instance of the Form object.
- WindowContainer is compatible with both the Form and the MDIForm classes.

WindowContainer Class



Single Window Persistent

- Below is a screen shot of a single Window consisting of one non-persistent procedure and three persistent procedures (studbr.p):



Single Window Persistent

```
on value-changed of b1
run openq in barhandle (buffer student).
open query q1 for each student.
enable b1 with frame f1.
```

- Insert the embedded form logic after the last trigger and before the first time through logic before the Wait-For.
- Ensures the GUI window is attached to the form before the window is realized

Single Window Persistent

- Define the following variables:

```
/* mainform gui window contains the wincontainer */
DEFINE VARIABLE MainForm AS Progress.Windows.Form.

/* wincontainer contains the progress gui window */
DEFINE VARIABLE WinContainer AS
Progress.Windows.WindowContainer NO-UNDO.

/* hwin is handle for dynamic progress window */
DEFINE VARIABLE hWin AS HANDLE NO-UNDO.
```

Single Window Persistent

- Since the default-window is already realized, we need to create a window and assign it to the current window.
- The current-window is where all the unrealized frames are displayed when they are brought into view.

```
CREATE WINDOW hWin ASSIGN
   WIDTH = 120
   HEIGHT = 20.
current-window = hwin.
```

Single Window Persistent

- The client area for the Form is set to the size of the ABL Window.

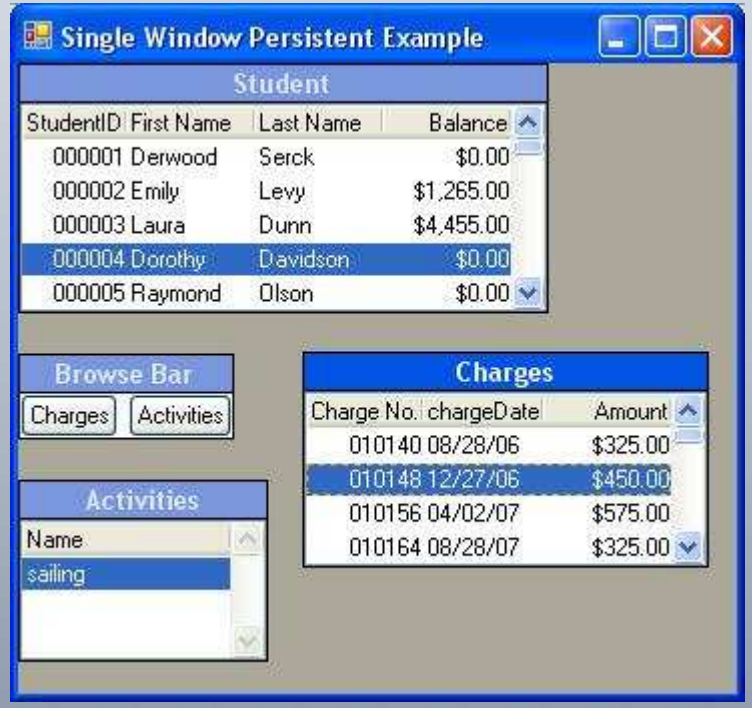
```
/* Create the form. */
MainForm = NEW Progress.Windows.Form( ).
MainForm:Text = “Single Window Persistent
Example".
MainForm:ClientSize = NEW Size(hwin:WIDTH-
PIXELS, hwin:HEIGHT-PIXELS).
MainForm:Show( ).
```

Single Window Persistent

```
/* Create the WindowContainer, embedding the window
into it. Set size to the ABL Window size. */

WinContainer = NEW Progress.Windows.WindowContainer( ).

WinContainer:Size = NEW Size( hwin:WIDTH-PIXELS,

hwin:HEIGHT-PIXELS).

WinContainer:Location = new Point(0,0).

WinContainer:EmbeddedWindow = hwin.

WinContainer:Parent = MainForm.

WinContainer:Show( ).

view hwin.
```

Single Window Persistent

- Notice the Form looks almost identical to the ABL Window. The icon symbol appears to be the only difference (studbrnet.p).



MakeForm.cls

- To automate attaching GUI windows in our application to .NET Forms, the MakeForm class is created.
- The MakeForm class receives the title string, window handle, the starting vertical position and passes back the MainForm instance.
- The MakeForm class contains the same code that was in the studbrnet.p.
- MakeForm.cls will be used in the rest of the examples in this presentation.

MakeForm.cls

```
define var makeformvar as makeform.
define var MainForm    as Progress.Windows.Form.

CREATE WINDOW hWin ASSIGN
  WIDTH = 120
  HEIGHT = 20.
current-window = hwin.
makeformvar = new makeform(input "ADM2 Page
     Form", input hwin, input 0, output MainForm).
view hwin.
```

ADM2 to .NET

- MakeForm.cls may also be used for SMARTContainers in ADM2.
- The include file: windowmn.i must be modified to include the corresponding Form wait-for:

  wait-for System.Windows.Forms.Application:Run(MainForm).
- The include file name windowmnnet.i contains the above wait-for statement and is used instead of windowmn.i in (wpagefoldernet2.w) .

ADM2 to .NET

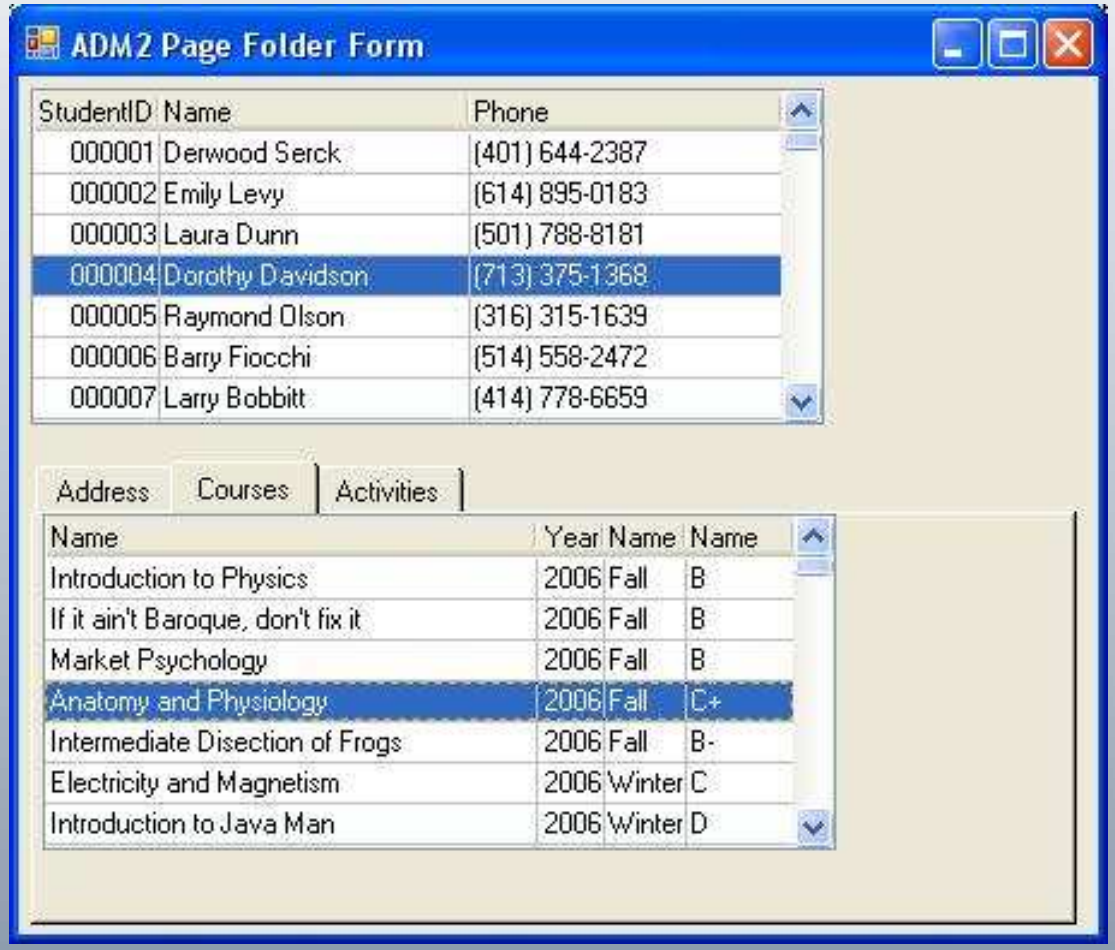
- Put in definitions section:

```
define var makeformvar as makeform.

DEFINE VARIABLE MainForm AS Progress.Windows.Form

NO-UNDO.
```

- Put in main section:

```
makeformvar = new makeform(input "ADM2 Page Form", input
  wwin, input 0, output MainForm).

/* Include custom  Main Block code for SmartWindows. */

{windowmnnet.i}
```

ADM2 to .NET



Recursive Menus

- The ABL Dynamic Menu Program uses recursive menus with the following table:

Menu

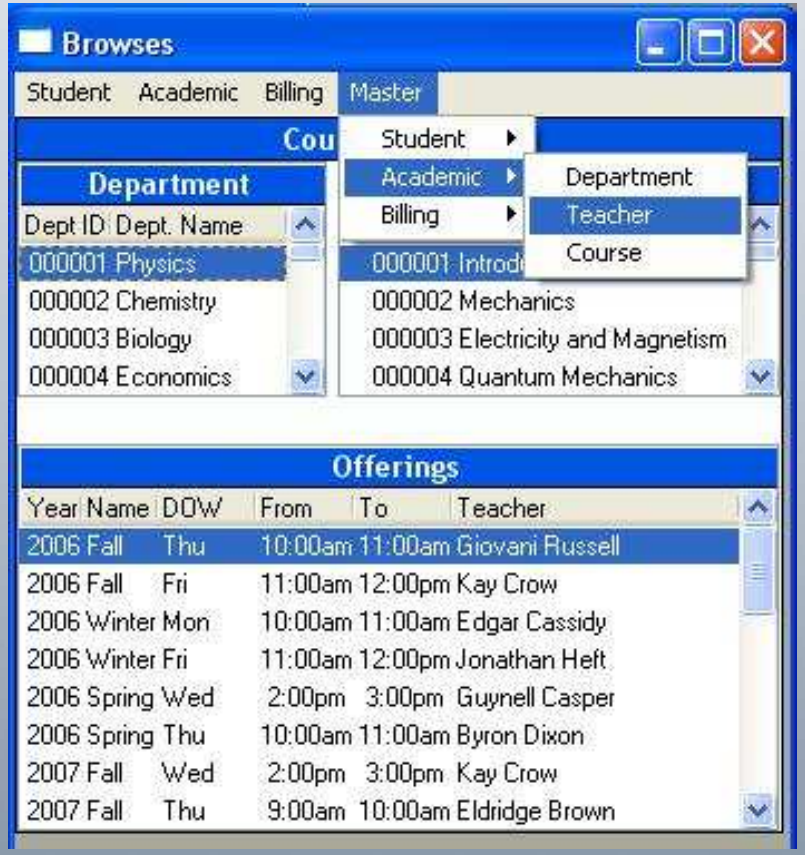
Menu Number	Menu Number Above	Menu Name	Menu Type	Order Number	Program Name
PK, SA	FK	NN	NN	NN	NN
1		Browses	MB	10	
2	1	Student	MI	10	ppstudent.p
3	1	Academic	SM	10	
4	3	Offering	MI	10	ppoffering.p
5	3	Registration	MI	20	ppregistration.p

Recursive Menus

- In dynmenu**1**.p, the buildmenu internal procedure is called recursively, by reading all the menu records for a particular menu number above (menu-no-above).
- If the menu type code (m-type-code) is a sub-menu ("SM") then a sub-menu widget is created and parented to the menu handle and buildmenu is called with that submenu being the menu above.
- If the menu type code is a menu-item ("MI") then a menu-item widget is created and parented to the menu handle and a temp-table record is created containing the menu-name and the run program.

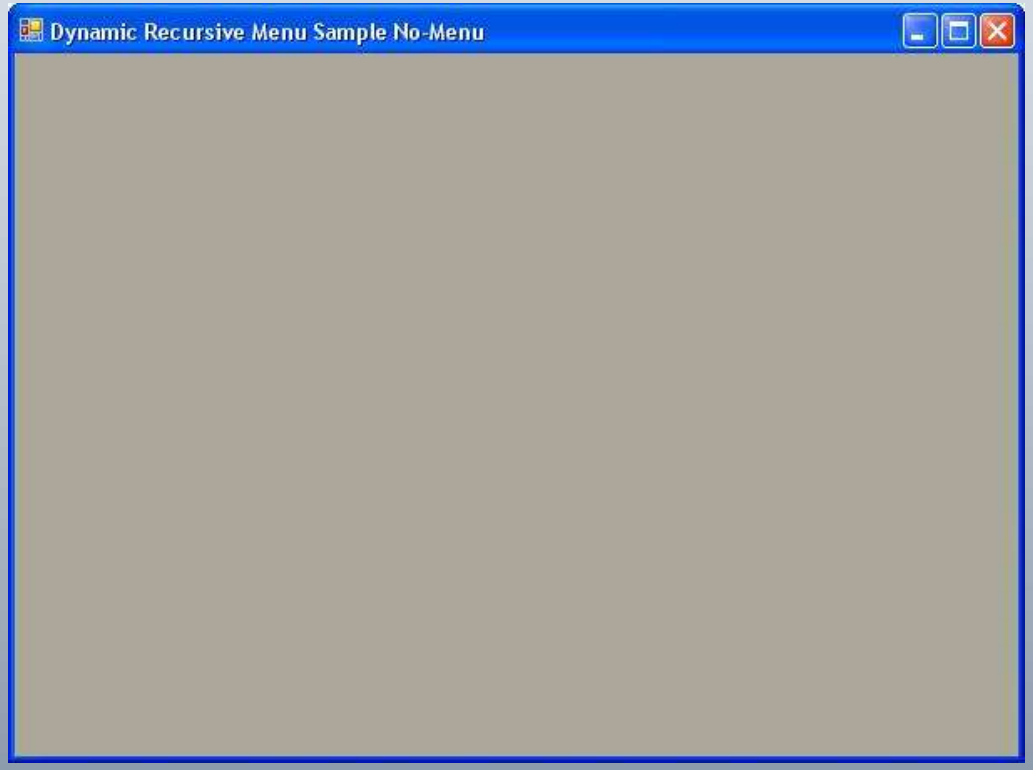
Menus to Toolbars

- What happens when there is a menubar attached to an ABL Window and it is embedded in a Form?



Menus to Toolbars

- The result is a Form with no menubar! Menu and submenu widgets are not visible in a form when the ABL Window is embedded (dynmenu**1**a.p) .



UltraToolBar Components

- UltraToolBarsManager – This control manages all ToolBars and Tools that are attached to it.
  - The UltraToolBarsManager instance should be docked within a container using the DockWithinContainer property for the UltraToolBarsManager.
- UltraToolBar – This toolbar has three types:
  - Standard
  - Task Pane Toolbar
  - Main Menu Bar
- This presentation focuses on the Main Menu Bar.

UltraToolBar Components

- Toolbar Tools – The following are the various types of tools (this presentation will focus on Buttons and Popup Menus):

▪ Button	▪ ComboBox	▪ Control Container
▪ Font List	▪ Label	▪ Masked Edit
▪ Popup Color Picker	▪ Popup Control Container	▪ Popup Gallery
▪ Popup Menu	▪ Progress Bar	▪ State Button
▪ TextBox		

UltraToolBar Vocabulary

- Here is a table showing a mapping of the ABL menu widget to the equivalent GUI for .NET control:

ABL GUI Widget	GUI for .NET Control
Menu (Pull Down or Pop-Up)	UltraToolBar (Main Menu Bar or Standard)
Sub-Menu	Tool of type Popup Menu
Menu-Item	Tool of type Button

UltraToolBar Steps

- Create instances for the UltraToolBarsManager and the UltraToolBar being used for the menu bar (dynmenu**2**.p) .

```
utbmgr1 = new UltraToolbarsManager().

utb1 = new UltraToolbar(menu-name).
```

- Subscribe the ToolClick event to the internal procedure ToolClick.

```
utbmgr1:ToolClick:Subscribe("ToolClick").
```

UltraToolBar Steps (cont.)

- Set the UltraToolBar MainMenuBar property to yes.
- Add the UltraToolBar to the UltraToolBarsManager's Toolbars collection.

```
utbmgr1:ToolBars:Add(utb1).
```

- **Dock the UltraToolBarsManager with the MainForm instance.**

```
utbmgr1:DockWithinContainer = MainForm
```

- **Recursively read menu table**

UltraToolBar Steps (cont.)

```
PROCEDURE buildmenunet:

  def input parameter ipmenu-no as integer          no-undo.

  def input parameter iputb     as UltraToolBar     no-undo.

  def input parameter ipmenu    as PopupMenuTool    no-undo.

  def input parameter ipobjtype as char             no-undo.

  for each menu where menu-no-above = ipmenu-no:

   .

   .

   run buildmenunet

       (input menu-no, input ?, input pumt1,
              input"PopupMenuTool").

  end. /* for each */

end. /* procedure */
```

UltraToolBar Steps (cont.)

- Regardless whether the menu type is either a sub-menu or a menu-item:
  - Create a new PopupMenuTool or ButtonTool Instance
  - Set the caption to the menu-name
  - Add the tool to the UltraToolBarsManager Tools collection.
- Attach the newly created tool instance to appropriate parent control, either the UltraToolBar or the PopUp Menu above the tool.

UltraToolBar Steps (cont.)

```
when "MI" then do:

   tmpbt = new ButtonTool(menu-name +
    string(iseq)).

   tmpbt:SharedPropsInternal:Caption = menu-name.

   utbmgr1:Tools:add(tmpbt).

end.

.

.

when "PopupMenuTool" then do:

    ipmenu:Tools:add(tmpbt).

end.
```

UltraToolBar Steps (cont.)

- How does the Button tool call the correct program?
- The toolclick event can only be subscribed on the UltraToolBarsManager level.
- The toolclick event procedure uses the ToolClickEventArgs input parameter.
- The choosemi procedure has been subscribed by this-procedure.

```
subscribe "choosemi" in this-procedure.
```

- The toolclick event procedure publishes "choosemi" and passes the caption of the selected tool to choosemi.

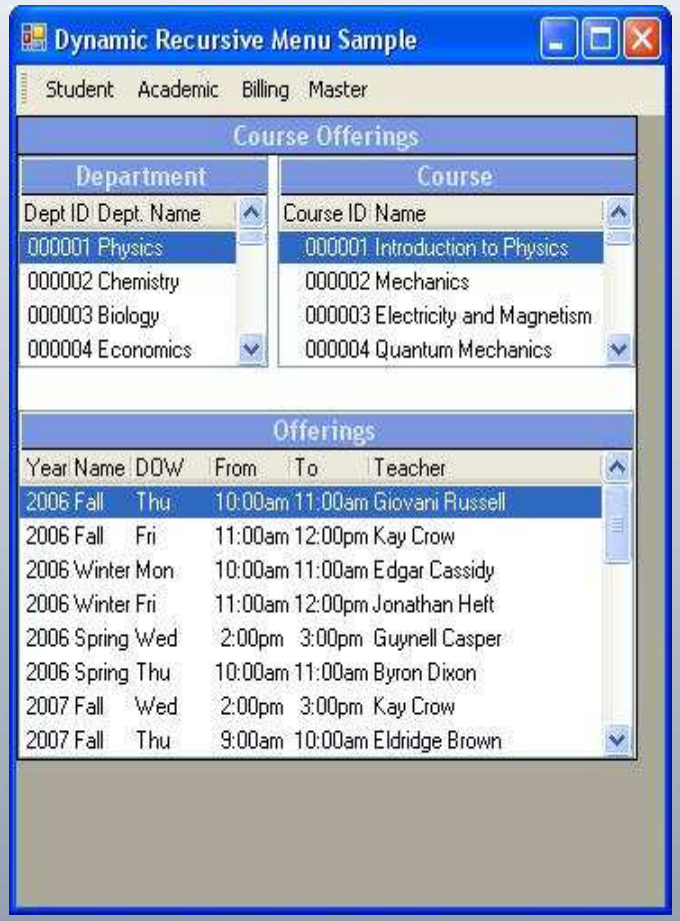
```
publish "choosemi"

(input e:tool:SharedPropsInternal:Caption).
```

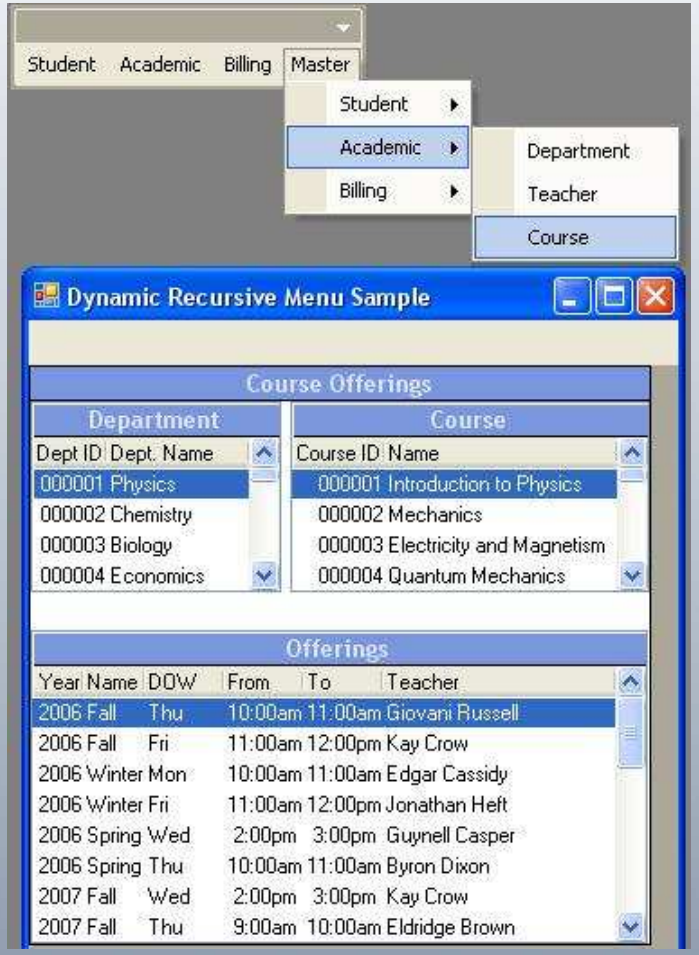
UltraToolBar Steps (cont.)

- The choosemi procedure looks up the menu name in a temp-table in order to execute the correct run program, just like it did the ABL GUI.
- In order to avoid the encroachment of the first row of the client area on the UltraToolBar at the top of the Form, the vertical position of the WinContainer's location is lowered 25 pixels from the top.

```
makeformvar = new makeform(input
     "Dynamic Recursive Menu Sample",
     input hwin, input 25, output
     MainForm).
```

UltraToolBar in Action





Belly up to the UltraToolBar and
Order from the Menu

Calling .NET Forms

- The next step is allow the menu system to call GUI for .NET forms.
- In dynmenu3.p, the tforminstance field of type Progress.Lang.Object is added to the tprogram temp-table.
- Similar to the thnd field for storing the current object instance.
- The choosemi procedure is modified to handle .NET form instances.

Calling .NET Forms (cont.)

```
if substr(tprogram.tproc,length(tprogram.tproc)- 3)
=    ".cls" then do:

classname = substr(tprogram.tproc,1,length
     (tprogram.tproc) - 4).

programform =
cast(tprogram.tforminstance,"Progress.Windows.Form")
no-error.
```

- A procedure that ends with .cls has its tforminstance field stored in the programform variable.

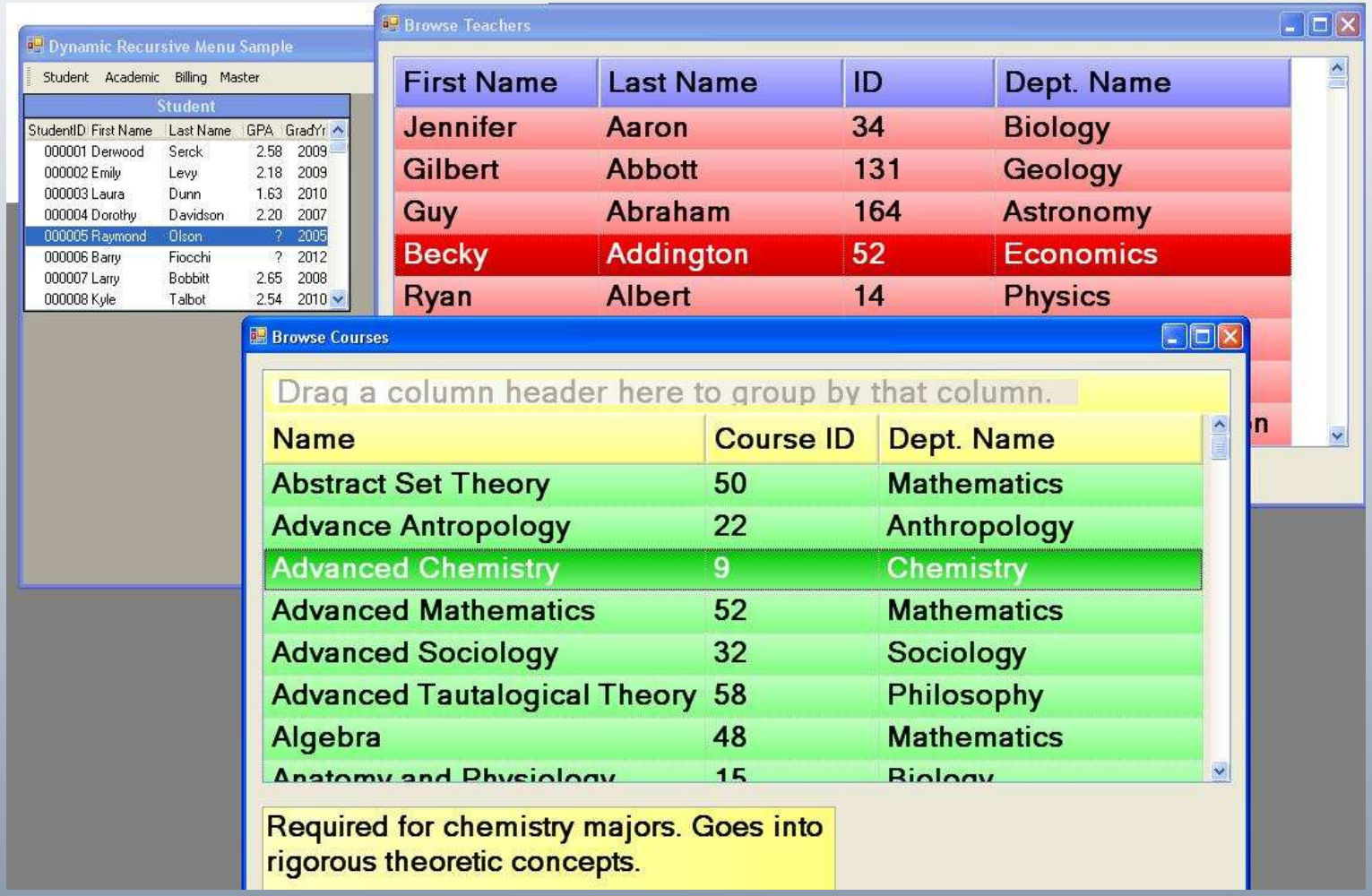
Calling .NET Forms (cont.)


```
if not valid-object(programform)
or
cast(tforminstance,"Progress.Windows.Form"):IsDisposed
then do:
        programform = dynamic-new classname () no-error.
        programform:show().
        tprogram.tforminstance = programform.
end.
else programform:activate().
```

Calling .NET Forms (cont.)

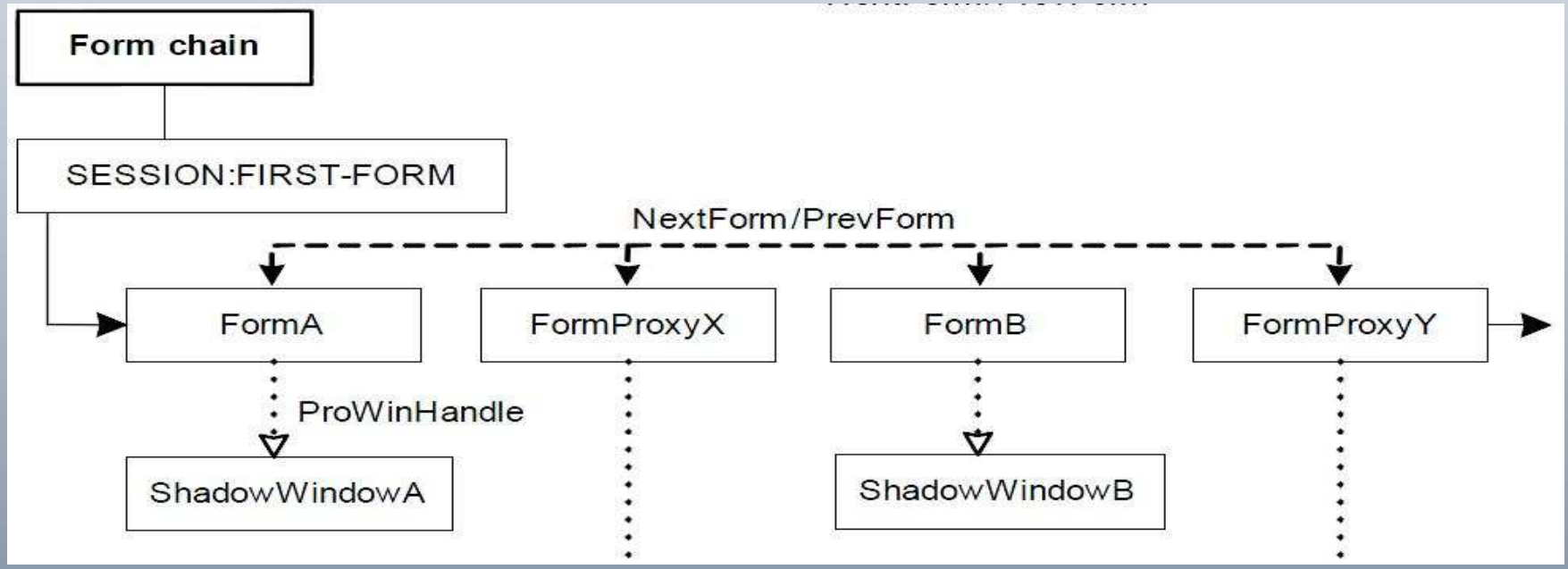
- Objective is to check if instance is current. If not create new instance otherwise bring the form to the front with the activate() method.
- IsDisposed property means the form has been closed.
- Create new instance using the dynamic-new function and store it in the tforminstance field.
- The show() method brings the form into view.

Calling .NET Forms (cont.)



Walking the Form Tree

- Progress provides a double linked list of forms similar to widgets.
- A form proxy is used to access GUI Windows in the same list.



Walking the Form Tree (cont.)

```
rIForm = SESSION:FIRST-FORM.
DO WHILE VALID-OBJECT( rIForm ):
  rForm = CAST(rIForm, Progress.Windows.Form) no-error.
  IF rForm <> ? THEN do:
        rformstr = string(rform).
        MESSAGE "Object Class:" entry(1,rFormstr) skip(1)
          "           " entry(2,rformstr)
          VIEW-AS ALERT-BOX INFORMATION.
  end.
  else do:
    rProxy = CAST(rIForm, Progress.Windows.FormProxy) no-error.
    IF rProxy <> ? THEN do:
      hwin = rProxy:ProWinHandle.
      MESSAGE "Object Class:" rProxy skip(1)
                  "  Gui Window:" hwin:title skip
            VIEW-AS ALERT-BOX INFORMATION
    end.
    rIForm = rIForm:NextForm.
  end.
end.
```

Calling User Controls

- The UltraToolBar may also call User Controls that may be attached to the main form.
- This is similar to using a GUI frame rather than a window.
- The UserControl class has both hide() and show() methods making it easy to manipulate.

Calling User Controls (cont.)

- In dynmenu4.p, the WindowContainer that contains the GUI window must be hidden. The hidewincontainer method that is part of the makeform class is used for this.
- After determining that the program is a class, the type-of function is used to determine if the class is a Form or a UserControl.
- The currentcontrol variable is analogous to the curh (current handle) variable on the Progress GUI side. It stores the currently displayed UserControl instance and is hidden when switching to another UserControl.

Calling User Controls (cont.)

```
currentcontrol =
cast(tprogram.tforminstance,"Progress.Windows.UserControl"
   ) no-error.
      makeformvar:hidewincontainer().
      if not valid-object(currentcontrol) then do:
        currentcontrol:dispose() no-error.
        currentcontrol =
  cast(plo,"Progress.Windows.UserControl") no-error.
        mainform:controls:add(currentcontrol).
        currentcontrol:Location = new Point(0,25).
        currentcontrol:show().
        tprogram.tforminstance = plo.
      end.
      else currentcontrol:show().
```

Calling User Controls (cont.)

Summary

- OpenEdge provides many options on integrating and migrating an ABL application to a GUI for .NET application.
- The UltraToolBar is a powerful control that can be used to select functions in a variety of ways in an application.

Questions